An Order-Theoretic Analysis of Universe Polymorphism

KUEN-BANG HOU (FAVONIA), University of Minnesota, USA
CARLO ANGIULI, Carnegie Mellon University, USA
REED MULLANIX, University of Minnesota, USA

We present a novel formulation of universe polymorphism in dependent type theory in terms of monads on the category of strict partial orders, and a novel algebraic structure, *displacement algebras*, on top of which one can implement a generalized form of McBride's "crude but effective stratification" scheme for lightweight universe polymorphism. We give some examples of exotic but consistent universe hierarchies, and prove that every universe hierarchy in our sense can be embedded in a displacement algebra and hence implemented via our generalization of McBride's scheme. Many of our technical results are mechanized in Agda, and we have an OCaml library for universe levels based on displacement algebras, for use in proof assistant implementations.

CCS Concepts: • **Theory of computation** → *Proof theory*; **Type theory**; Lambda calculus.

Additional Key Words and Phrases: type theory, universes, universe polymorphism

ACM Reference Format:
Kuen-Bang Hou (Favonia), Carlo Angiuli, and Reed Mullanix. 2023. An Order-Theoretic Analysis of Universe Polymorphism. *Proc. ACM Program. Lang.* 7, POPL, Article 57 (January 2023), 27 pages. https://doi.org/10.1145/3571250

1 INTRODUCTION

Dependent type theories are popular frameworks for mechanized proofs in which types can be manipulated as terms. In dependent type theory, a simple way to represent indexed families of types is to consider functions into a *universe type* (written `Type`, `Set`, $\mathfrak{U}$, etc.) whose elements are types. Early dependent type theories assumed that the type universe is an element of itself $\mathfrak{U} : \mathfrak{U}$ [Martin-Löf 1971], but this was soon shown by Girard to be inconsistent [Coquand 1986]. Martin-Löf [1975] subsequently considered *stratified* collections of universes, in which smaller universes are elements of larger universes but not vice versa. Such *universe hierarchies* are indexed by universe levels (often the natural numbers), which are the subject of this paper.

Passing from a single universe to a universe hierarchy solves the consistency problem, but it results in usability headaches: suddenly, definitions involving universes must be replicated across many universe levels. To define even something as simple as the polymorphic identity function $\forall \mathsf{a}.\mathsf{a} \to \mathsf{a}$, we must fix the type over which a ranges. Suppose we choose the smallest universe $\mathfrak{U}_0$:

$$\mathtt{id} : \textstyle\prod_{A:\mathfrak{U}_0} (A \to A)$$

Authors' addresses: Kuen-Bang Hou (Favonia), kbh@umn.edu, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, Minnesota, 55455, USA; Carlo Angiuli, cangiuli@cs.cmu.edu, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, Pennsylvania, 15213, USA; Reed Mullanix, rmullani@umn.edu, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, Minnesota, 55455, USA.

2475-1421/2023/1-ART57
https://doi.org/10.1145/3571250

Then we can instantiate `id` at any element of $\mathfrak{U}_0$, such as `bool`, `int`, `int → int`, …, but not $\mathfrak{U}_0$ itself (or $\mathfrak{U}_0 \to \mathfrak{U}_0$, etc.), because we do not have $\mathfrak{U}_0 : \mathfrak{U}_0$! We can define a larger copy of `id`,

$$\texttt{id}' : \textstyle\prod_{A:\mathfrak{U}_1} (A \to A)$$

and this can be instantiated at `bool`, $\mathfrak{U}_0$, $\mathfrak{U}_0 \to \mathfrak{U}_0$, …, but not $\mathfrak{U}_1$, *ad infinitum*.

Many systems address this problem by adopting one of a number of extensions to type theory that allow users to write *universe-polymorphic* definitions that can be uniformly instantiated at various universe levels. There are at least four well-known mechanisms for universe polymorphism:

Explicit level quantification. Systems such as Agda [The Agda Development Team 2022] and Lean [de Moura et al. 2015] index universes by a special type/sort of `Levels` defined as a join semilattice generated by level variables and the operations `zero`, `suc`, and `max` satisfying various equations. Users can write explicitly polymorphic definitions that quantify over levels; in Agda, for example, one writes $\texttt{id} : \prod_{\ell:\texttt{Level}} \prod_{A:\mathfrak{U}_\ell} (A \to A)$.

Explicit user constraints. In Matita [The HELM Team 2016], users can declare universes and impose constraints between them, and a type checker determines whether these constraints are sufficient. Here is some example code from the Matita library:

```
universe constraint Type[0] < Type[1].
universe constraint Type[1] < Type[2].
```

Typical ambiguity. Users of systems with *typical ambiguity* do not need to specify the levels of universes; they may innocently pretend that $\mathfrak{U} : \mathfrak{U}$, and their proof assistant will use constraint solving to determine a consistent assignment of universe levels, or reject the definition in the rare case that no such assignment exists. Algorithms for typical ambiguity were originally developed by Huet [1987] and Harper and Pollack [1991] and implemented in LEGO [The LEGO Team 1999], Coq [Sozeau and Tabareau 2014; The Coq Development Team 2022], and Idris [Brady 2013] among others.

Crude but effective stratification. McBride has proposed a simple form of universe polymorphism in which every judgment is uniformly invariant under incrementing all universe levels [McBride 2002, 2011]. A *displacement* operation $(-)^{\Uparrow^n}$ takes in closed terms and raises all universe levels inside them by a fixed natural number n. Under this scheme, users can therefore write apparently non-universe-polymorphic functions such as $\texttt{id} : \prod_{A:\mathfrak{U}_0} (A \to A)$, and apply them at higher universe levels after the fact. For example, if we want to apply `id` to a type not in $\mathfrak{U}_0$, such as $\mathfrak{U}_0$ itself (an element of $\mathfrak{U}_1$), we can displace `id` by one level by writing $\texttt{id}^{\Uparrow^1} : \prod_{A:\mathfrak{U}_1} (A \to A)$, obtaining the desired application $\texttt{id}^{\Uparrow^1}(\mathfrak{U}_0) : \mathfrak{U}_0 \to \mathfrak{U}_0$.

In our work on various proof assistant implementations, we have come to appreciate McBride's "crude but effective stratification" for its surprising efficacy and simplicity: users can avoid explicit level quantification, and implementers can avoid solving inequality constraints involving joins (maximums) or successors of universe levels. This led us to wonder how general McBride's mechanism truly is; as this paper will show, it is quite general indeed.

Contributions. Our main contributions are a novel formulation of universe polymorphism in terms of monads over strict partial orders, and a novel algebraic structure, *displacement algebras*, sufficient to operate "crude but effective" universe polymorphism. We then prove that any universe-polymorphic type theory can be phrased (in an appropriate sense) as a generalized form of "crude but effective stratification" by choosing a matching displacement algebra.

In Section 2, we present our new monadic description of universe-polymorphic type theory. In Section 3.1, we introduce our notion of displacement algebra, and formulate our generalization of "crude but effective stratification"; in Section 3.3 we present a variety of interesting displacement algebras. In Section 3.4 we construct a faithful functor sending any system of universe polymorphism

to a "crude but effective" displacement system, under mild hypotheses. In Section 4 we discuss our Agda formalization of many of our technical results, as well as our OCaml library mugen for universe levels using displacement algebras [RedPRL Development Team 2022a]. We are currently using mugen in the experimental proof assistant `algaett` [RedPRL Development Team 2022b].

Non-Contributions. There are two important aspects of universe hierarchies that we will not address in this paper: cumulativity and kinds. A universe hierarchy is *cumulative* if an element of a smaller universe can be *implicitly* coerced to an element of a larger one; i.e. if smaller universes are subtypes of larger ones. Coercions between universes obey different equations in different systems; we consider explicit (but strict) coercions that can be considered as an elaboration of cumulativity. Most proof assistants support cumulativity with the notable exception of Agda, in part because it supports universe polymorphism outside the prenex fragment. We consider only prenex level quantification, effectively stratifying levels and terms, which avoids the difficulties Agda faces.

The *kind* of a universe refers to the structure(s) shared by its elements; for example, one might consider a universe of strict propositions [Gilbert et al. 2019], types whose elements are all definitionally equal. The universe hierarchy would then be enriched with these universe kinds. In this paper, we focus only on universe levels because the choice of levels is frequently orthogonal to the kind system; most type checkers can work with an arbitrary decidable partial order on levels and perhaps a successor operator, without knowing how levels are represented. We will describe how kinds may be integrated into our framework in Section 5.1.

2 GENERALIZED UNIVERSE HIERARCHIES

We start with the syntax of dependent type theory with a generalized notion of universe hierarchy.

2.1 Universe-Monomorphic Type Theory

When formulating the syntax of a core dependent type theory with a hierarchy of type universes, one often indexes the universes by (an external copy of) the natural numbers $\mathbb{N}$. Such a universe hierarchy is characterized by several key properties:

(1) Elements of a universe $A : \mathfrak{U}_i$ can be regarded as types, whether directly ("universes à la Russell") or via an explicit coercion function $\mathrm{El}^i(A)$ ("universes à la Tarski").
(2) Each universe is closed under (some or all of) the type formers considered in the language: for example, if $A : \mathfrak{U}_i$ and $B : \mathrm{El}^i(A) \to \mathfrak{U}_i$ then $\prod_{x:A} B(x) : \mathfrak{U}_i$.
(3) Each universe is an element of the next universe: $\mathfrak{U}_0 : \mathfrak{U}_1 : \mathfrak{U}_2 : \ldots$. To avoid inconsistency, it is crucial that $\mathfrak{U}_i$ is *not* an element of itself.
(4) Every $A : \mathfrak{U}_i$ is also an element of $\mathfrak{U}_{i+1}$, either directly ("cumulativity") or via an explicit *lifting* operation $\Uparrow_i^{i+1}(A) : \mathfrak{U}_{i+1}$.

For the moment, we are considering a *universe-monomorphic* system without variable universe levels, which will not appear until in Section 2.2.

We begin by considering the syntax of *L-monomorphic type theory*: a dependent type theory with a universe hierarchy indexed not by $\mathbb{N}$ but an arbitrary (fixed, external) poset $(L, \leq_L)$ of universe level expressions, where properties (3) and (4) must be rephrased in terms of $\leq_L$:

(3′) Whenever $\ell' <_L \ell$, we have $\mathfrak{U}_{\ell'} : \mathfrak{U}_\ell$.
(4′) Whenever $\ell' \leq_L \ell$, if $A : \mathfrak{U}_{\ell'}$ then $\Uparrow_{\ell'}^{\ell}(A) : \mathfrak{U}_\ell$.

In (3′), $<_L$ is the *strict* order associated to $\leq_L$, namely $\ell' <_L \ell$ when $\ell' \leq_L \ell$ and $\ell' \neq \ell$. Notably, we do *not* require that $<_L$ be well-founded.

Notation 2.1. For a poset $(L, \leq_L)$ we write $L^\top$ for the poset that adjoins a top element $\top$ to L: that is, the set $L \coprod \{\top\}$ with the partial order that extends $\leq_L$ with $\ell <_{L^\top} \top$ for $\ell \in L$.

$$\begin{array}{rl} \text{Contexts} & \Gamma ::= \cdot \mid \Gamma, x{:}A \\ \text{Types} & A, B ::= \mathrm{El}^{\ell}(e) \mid \Uparrow^{\ell}_{\ell'}(A) \mid \prod^{\ell}_{x:A} B \mid \mathbf{0}^{\ell} \mid \mathfrak{U}^{\ell}_{\ell'} \\ \text{Terms} & e, e' ::= x \mid \mathrm{code}^{\ell}(A) \mid \lambda(x{:}A).e \mid e(e') \mid \mathrm{abort}_{x.A}(e) \end{array}$$

$$\frac{}{\cdot\ \mathrm{ctx}_L} \qquad \frac{\Gamma\ \mathrm{ctx}_L \qquad \Gamma \vdash_L A\ \mathrm{type}_\top \qquad x \notin \Gamma}{\Gamma, x{:}A\ \mathrm{ctx}_L} \qquad \frac{\ell \in L \qquad \Gamma \vdash_L e : \mathfrak{U}^{\top}_{\ell}}{\Gamma \vdash_L \mathrm{El}^{\ell}(e)\ \mathrm{type}_\ell}$$

$$\frac{\ell' \leq_{L^\top} \ell \qquad \Gamma \vdash_L A\ \mathrm{type}_{\ell'}}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell'}(A)\ \mathrm{type}_\ell} \qquad \frac{\Gamma \vdash_L A\ \mathrm{type}_\ell}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell}(A) \equiv A\ \mathrm{type}_\ell} \qquad \frac{\ell'' \leq_{L^\top} \ell' \leq_{L^\top} \ell \qquad \Gamma \vdash_L A\ \mathrm{type}_{\ell''}}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell'}(\Uparrow^{\ell'}_{\ell''}(A)) \equiv \Uparrow^{\ell}_{\ell''}(A)\ \mathrm{type}_\ell}$$

$$\frac{\Gamma \vdash_L A\ \mathrm{type}_\ell \qquad \Gamma, x{:}\Uparrow^{\top}_{\ell}(A) \vdash_L B\ \mathrm{type}_\ell}{\Gamma \vdash_L \prod^{\ell}_{x:A} B\ \mathrm{type}_\ell} \qquad \frac{\ell' \leq_{L^\top} \ell \qquad \Gamma \vdash_L A\ \mathrm{type}_{\ell'} \qquad \Gamma, x{:}\Uparrow^{\top}_{\ell'}(A) \vdash_L B\ \mathrm{type}_{\ell'}}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell'}(\prod^{\ell'}_{x:A} B) \equiv \prod^{\ell}_{x:\Uparrow^{\ell}_{\ell'}(A)} \Uparrow^{\ell}_{\ell'}(B)\ \mathrm{type}_\ell}$$

$$\frac{}{\Gamma \vdash_L \mathbf{0}^{\ell}\ \mathrm{type}_\ell} \qquad \frac{\ell' \leq_{L^\top} \ell}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell'}(\mathbf{0}^{\ell'}) \equiv \mathbf{0}^{\ell}\ \mathrm{type}_\ell} \qquad \frac{\ell' <_{L^\top} \ell}{\Gamma \vdash_L \mathfrak{U}^{\ell}_{\ell'}\ \mathrm{type}_\ell} \qquad \frac{\ell'' <_{L^\top} \ell' \leq_{L^\top} \ell}{\Gamma \vdash_L \Uparrow^{\ell}_{\ell'}(\mathfrak{U}^{\ell'}_{\ell''}) \equiv \mathfrak{U}^{\ell}_{\ell''}\ \mathrm{type}_\ell}$$

$$\frac{\Gamma\ \mathrm{ctx}_L \qquad \Gamma(x) = A}{\Gamma \vdash_L x : A} \qquad \frac{\ell \in L \qquad \Gamma \vdash_L A\ \mathrm{type}_\ell}{\Gamma \vdash_L \mathrm{code}^{\ell}(A) : \mathfrak{U}^{\top}_{\ell}} \qquad \frac{\ell \in L \qquad \Gamma \vdash_L A\ \mathrm{type}_\ell}{\Gamma \vdash_L \mathrm{El}^{\ell}(\mathrm{code}^{\ell}(A)) \equiv A\ \mathrm{type}_\ell}$$

$$\frac{\ell \in L \qquad \Gamma \vdash_L e : \mathfrak{U}^{\top}_{\ell}}{\Gamma \vdash_L e \equiv \mathrm{code}^{\ell}(\mathrm{El}^{\ell}(e)) : \mathfrak{U}^{\top}_{\ell}} \qquad \frac{\Gamma, x{:}A \vdash_L e : B}{\Gamma \vdash_L \lambda(x{:}A).e : \prod^{\top}_{x:A} B} \qquad \frac{\Gamma \vdash_L e_1 : \prod^{\top}_{x:A} B \qquad \Gamma \vdash_L e_2 : A}{\Gamma \vdash_L e_1(e_2) : B[x \mapsto e_2]}$$

$$\frac{\Gamma, x{:}A \vdash_L e_1 : B \qquad \Gamma \vdash_L e_2 : A}{\Gamma \vdash_L (\lambda(x{:}A).e_1)(e_2) \equiv e_1[x \mapsto e_2] : B[x \mapsto e_2]} \qquad \frac{\Gamma \vdash_L e : \prod^{\top}_{x:A} B}{\Gamma \vdash_L e \equiv \lambda(x{:}A).e(x) : \prod^{\top}_{x:A} B}$$

$$\frac{\Gamma, x{:}\mathbf{0}^{\top} \vdash_L A\ \mathrm{type}_\top \qquad \Gamma \vdash_L e : \mathbf{0}^{\top}}{\Gamma \vdash_L \mathrm{abort}_{x.A}(e) : A[x \mapsto e]}$$

Fig. 1. Selected rules of L-monomorphic type theory for a poset L.

We present the interesting inference rules of L-monomorphic type theory in Figure 1 (omitting standard rules of equality, e.g. congruence and conversion); for brevity, we consider a type theory with only Π-types, the empty type, and universe types.[1] We annotate each judgment with the poset L because we will soon need to work with many different choices of L at the same time; however, all the rules of Figure 1 leave L fixed.

Notably, we present our universes "à la Coquand" [Coquand 2013, 2019; Gratzer et al. 2020], in which the type judgment is indexed by either a universe level or $\top$. Here $A\ \mathrm{type}_\top$ corresponds to the ordinary A type judgment (in particular, all types A appearing in contexts or to the right of the colon are $A\ \mathrm{type}_\top$), and for $\ell \neq \top$ the collection of types A in $A\ \mathrm{type}_\ell$ is isomorphic to the collection of elements of $\mathfrak{U}^{\top}_{\ell}$ (via El^{ℓ} and code^{ℓ}). We write $\mathfrak{U}^{\ell'}_{\ell}$ for the code for type_ℓ in $\mathrm{type}_{\ell'}$, and may omit the superscript $\top$ for brevity. This presentation has the benefit of requiring fewer rules: Π-types and

[1] The empty type is solely for consistency arguments in Section 2.3.

closure of universes under Π-types are expressed in a single rule, and the Tarski El^ℓ and universe lift $\mathfrak{U}_{\ell'} \to \mathfrak{U}_\ell$ operations are instances of a single lifting operation ($\Uparrow_\ell^\top$ and $\Uparrow_{\ell'}^\ell$ respectively).

The presuppositions of the judgments in Figure 1 are as follows:

- $\Gamma \vdash_L A\ \mathrm{type}_\ell$ presupposes $\ell \in L^\top$ and $\Gamma\ \mathrm{ctx}_L$.
- $\Gamma \vdash_L A \equiv B\ \mathrm{type}_\ell$ presupposes $\ell \in L^\top$ and $\Gamma\ \mathrm{ctx}_L$.
- $\Gamma \vdash_L e : A$ presupposes $\Gamma\ \mathrm{ctx}_L$ and $\Gamma \vdash_L A\ \mathrm{type}_\top$.
- $\Gamma \vdash_L e_1 \equiv e_2 : A$ presupposes $\Gamma\ \mathrm{ctx}_L$ and $\Gamma \vdash_L A\ \mathrm{type}_\top$.

These presuppositions are implicit additional premises to the rules above that we have omitted for the sake of readability; for example, any rule with a premise or conclusion of the form $\Gamma \vdash_L A\ \mathrm{type}_\ell$ should be understood as having implicit premises $\ell \in L^\top$ and $\Gamma\ \mathrm{ctx}_L$. The role of these premises is to ensure the invariant that whenever $\Gamma \vdash_L A\ \mathrm{type}_\ell$ is derivable, $\ell \in L^\top$ and $\Gamma\ \mathrm{ctx}_L$ are derivable.

Remark. *In Figure 1, we assume that lifts are strictly functorial and commute strictly with type formers; some type theories omit either or both of these principles for syntactic or semantic reasons. We include them simply to demonstrate that they pose no issues to our formulation of universe levels in terms of partial orders. Our use of "universes à la Coquand" is likewise inessential.*

Example 2.2 (Natural numbers). Setting $L = \mathbb{N}$ we recover the standard universe hierarchy, in which every universe is contained in a larger universe ($\mathfrak{U}_i : \mathfrak{U}_{i+1}$).

Example 2.3 (Integers). Setting $L = \mathbb{Z}$ we produce a type theory with a non-well-founded universe hierarchy: every universe is contained in a larger universe *and contains a smaller universe* ($\mathfrak{U}_{i-1} : \mathfrak{U}_i : \mathfrak{U}_{i+1}$). Surprisingly, $\mathbb{Z}$-indexed universes have some practical applications. In Section 1, we discussed how the universe-monomorphic $\prod_{A:\mathfrak{U}_0}(A \to A)$ is not a satisfactory type for `id` because it does not let us instantiate A with a universe. This is true for the usual $\mathbb{N}$-indexed universe hierarchies because $\mathfrak{U}_0$ contains no universes; however, in a type theory with a $\mathbb{Z}$-indexed hierarchy, we *can* instantiate A with $\mathfrak{U}_{-1}$, avoiding the need for universe polymorphism in this example!

One could also consider the opposite poset of Example 2.2, namely $L = \mathbb{Z}_{\leq 0}$, a non-well-founded hierarchy in which every universe contains a smaller one ($\mathfrak{U}_{i-1} : \mathfrak{U}_i$).

Example 2.4 (Rationals). Setting $L = \mathbb{Q}$ we produce a type theory in which between any two universes there is a third universe ($\mathfrak{U}_i : \mathfrak{U}_{(i+j)/2} : \mathfrak{U}_j$). As in Example 2.3, rational levels allow us to avoid some instances of universe polymorphism: if we have a term with the universe-polymorphic type $\prod_{A:\mathfrak{U}_\ell}\prod_{B:\mathfrak{U}_{\ell'}}\ldots$ for any levels $\ell' > \ell$, it is equally general in a $\mathbb{Q}$-indexed hierarchy to assign the monomorphic type $\prod_{A:\mathfrak{U}_0}\prod_{B:\mathfrak{U}_1}\ldots$ because arbitrarily many universes fit between $\mathfrak{U}_0$ and $\mathfrak{U}_1$.

$\mathbb{N}$-indexed universe hierarchies are usually formulated with the successor operation $\mathfrak{U}_i : \mathfrak{U}_{i+1}$ rather than $\mathfrak{U}_i : \mathfrak{U}_j$ for $i < j$; the former infers the most general type of the universe code $\mathfrak{U}_i$, whereas the latter checks the term $\mathfrak{U}_i$ against the type $\mathfrak{U}_j$. However, not all posets have a notion of successor, e.g. Example 2.4 above. Our rule must therefore be stated using the strict order $<_L$:

$$\textsc{standard}\quad \frac{\ell \in \mathbb{N}}{\Gamma \vdash_{\mathbb{N}} \mathfrak{U}_\ell\ \mathrm{type}_{\ell+1}} \qquad\qquad \textsc{our rule}\quad \frac{\ell' <_{L^\top} \ell}{\Gamma \vdash_L \mathfrak{U}_{\ell'}\ \mathrm{type}_\ell}$$

Derivations in L-monomorphic type theory depend on L only via $\ell, \ell' \in L$ and the relations $\ell' <_L \ell$ and $\ell' \leq_L \ell$. Therefore, any function $L \to L'$ that preserves $\ell' <_L \ell$ (and thus also $\ell' \leq_L \ell$) induces a transformation of derivations, judgments, and terms from L-monomorphic type theory into derivations, judgments, and terms in L'-monomorphic type theory. More precisely:

Definition 2.5. SOrd is the category of posets with <-preserving maps: functions $\sigma : L \to L'$ such that $x <_L y \implies \sigma(x) <_{L'} \sigma(y)$.

Remark. *$A <$-preserving map need not be a monomorphism (need not reflect equality), because it could identify unrelated points. For any poset L, the map $L \amalg L \to L$ that merges two copies of L into a single copy of L is not injective but nevertheless preserves the strict order.*

Notation 2.6. L-monomorphic type theory is functorial in L in the following sense. Let $\sigma : L \to L'$ in SOrd; we write $(-)[\sigma]$ for the level replacement function that maps judgments, derivations, contexts, types, and terms of L-monomorphic type theory to corresponding notions in L'-monomorphic type theory by sending levels $\ell \in L$ to $\sigma(\ell) \in L'$. For example, if $\mathcal{J}$ is a judgment in L-monomorphic type theory, $\mathcal{J}[\sigma]$ is a judgment in L'-monomorphic type theory.

Lemma 2.7. *Let $\sigma : L \to L'$ in* SOrd. *If a judgment $\mathcal{J}$ is derivable in L-monomorphic type theory, then the judgment $\mathcal{J}[\sigma]$ is derivable in L'-monomorphic type theory.*

Lemma 2.8. *Let $\sigma : L \to L'$ be a full*[2] *monomorphism in* SOrd. *Then a judgment $\mathcal{J}$ is derivable in L-monomorphic type theory if and only if $\mathcal{J}[\sigma]$ is derivable in L'-monomorphic type theory.*

Proof of Lemmas 2.7 and 2.8. By induction on derivations of $\mathcal{J}$. □

2.2 Universe-Polymorphic Type Theory

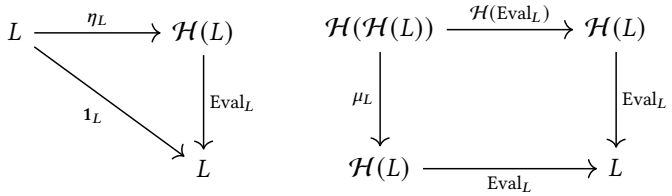
The essence of universe polymorphism is the ability to consider *variable* universe levels α, β that can be instantiated with concrete universe levels (such as natural numbers) or, depending on the system, other universe level expressions such as $\max(\alpha, \beta) + 1$. In short, each level context Δ gives rise to a universe hierarchy $\mathcal{H}(\Delta)$ of level expressions in context Δ, and thus a $\mathcal{H}(\Delta)$-monomorphic type theory (where the choice of $\mathcal{H}$ determines the valid expressions); substitutions between level contexts induce maps between monomorphic type theories (Lemma 2.7), and the collection of these type theories assembles into what we will call *$\mathcal{H}$-polymorphic type theory*.

What conditions are needed on $\mathcal{H}$ to realize this idea? First, the level expressions $\mathcal{H}(\Delta)$ must form a poset; traditionally Δ is a discrete poset (i.e. a set of unrelated level variables), but in our setting it is more natural to allow Δ to be any poset, such as $\alpha, \beta, \alpha < \beta$ (a context of two level variables with a strict inequality between them). $\mathcal{H}$ should preserve not only $\leq$ but also strict orders $<$; thus $\mathcal{H}$ is a functor SOrd $\to$ SOrd.

How do judgments at different level contexts interact? $\mathcal{H}$-polymorphic judgments in level context Δ correspond to $\mathcal{H}(\Delta)$-monomorphic judgments; analogously to ordinary substitutions, level substitutions from Δ to Δ' send level variables in Δ to level expressions in $\mathcal{H}(\Delta')$. Thus to transfer judgments from level context Δ ($\mathcal{H}(\Delta)$-monomorphic type theory) to Δ' using Lemma 2.7, we must lift maps $\Delta \to \mathcal{H}(\Delta')$ to $\mathcal{H}(\Delta) \to \mathcal{H}(\Delta')$. Combined with the requirement that level variables embed into level expressions ($\Delta \to \mathcal{H}(\Delta)$), this amounts to $\mathcal{H}$ being a monad:

Definition 2.9. A *hierarchy theory* $(\mathcal{H}, \eta, \mu)$ is a monad on SOrd.

Definition 2.10. Let $(\mathcal{H}, \eta, \mu)$ be a hierarchy theory, an *$\mathcal{H}$-algebra* is a pair of a poset L and a map $\mathrm{Eval}_L : \mathcal{H}(L) \to L$ in SOrd that commutes with η and μ as follows:



[2]Here we mean that σ is full as a functor $L \to L'$ between thin categories, i.e. that if $\sigma(\ell') \leq_{L'} \sigma(\ell)$ then $\ell' \leq_L \ell$.

Definition 2.11. For any hierarchy theory $(\mathcal{H}, \eta, \mu)$, we write $\mathrm{SOrd}^{\mathcal{H}}$ for the category of $\mathcal{H}$-algebras (i.e. the Eilenberg–Moore category of $\mathcal{H}$). The morphisms between two $\mathcal{H}$-algebras (L, Eval_L) and $(L', \mathrm{Eval}_{L'})$ are maps $\sigma : L \to L'$ that commute with the algebras:

$$\begin{array}{ccc} \mathcal{H}(L) & \xrightarrow{\mathcal{H}(\sigma)} & \mathcal{H}(L') \\ \downarrow{\scriptstyle \mathrm{Eval}_L} & & \downarrow{\scriptstyle \mathrm{Eval}_{L'}} \\ L & \xrightarrow[\sigma]{} & L' \end{array}$$

The Kleisli category is the full subcategory of the Eilenberg–Moore category $\mathrm{SOrd}^{\mathcal{H}}$ induced by *free $\mathcal{H}$-algebras* $(\mathcal{H}(\Delta), \mu_\Delta)$. Although we will restrict our attention to free $\mathcal{H}$-algebras, for technical convenience we will nevertheless work with the Eilenberg–Moore category rather than the Kleisli category of $\mathcal{H}$, because composition in the former coincides with composition in the base category SOrd, whereas composition in the Kleisli category is usually defined differently.

Notation 2.12. Let $(\mathcal{H}, \eta, \mu)$ be a hierarchy theory. We write $F^{\mathcal{H}}(\Delta) = (\mathcal{H}(\Delta), \mu_\Delta)$ for the free $\mathcal{H}$-algebra functor $\mathrm{SOrd} \to \mathrm{SOrd}^{\mathcal{H}}$ and $U^{\mathcal{H}}(\Delta, \mathrm{Eval}_\Delta) = \Delta$ for the forgetful functor $\mathrm{SOrd}^{\mathcal{H}} \to \mathrm{SOrd}$. The superscripts $\mathcal{H}$ in $F^{\mathcal{H}}$ and $U^{\mathcal{H}}$ may be omitted when they are clear from the context.

Notation 2.13. We write $|-|$ for the underlying carrier object of an algebra. For example, the underlying object of an $\mathcal{H}$-algebra is a poset.

The monadic structure indeed captures the expected substitution structure: an alternative way to present monads that is popular in Haskell involves "`return`" (the unit η) and "`bind`" (which is $\mu \circ \mathcal{H}(-)$); the $\mathtt{return} : \Delta \to \mathcal{H}(\Delta)$ stipulates that level variables are universe level expressions, and the $\mathtt{bind} : \mathcal{H}(\Delta) \to (\Delta \to \mathcal{H}(\Xi)) \to \mathcal{H}(\Xi)$ is the lifting of substitutions to functions between expressions. The monadic laws express the functionality and naturality of variable inclusion and substitution. The fact that the domain of $\mathcal{H}$ is SOrd rather than Set allows level contexts to specify (strict) inequalities, similarly to Matita. Therefore, Definition 2.9 precisely captures what we want.

There is one final subtlety before we present the syntax of $\mathcal{H}$-polymorphic type theory (for a fixed hierarchy theory $\mathcal{H}$). Recall that in functional programming languages, one can understand prenex type polymorphism as a way of "summarizing" some or all of a top-level definition's possible types. For instance, assigning the type $\forall \mathtt{a}.\mathtt{a} \to \mathtt{a}$ to $\mathtt{id} = \lambda \mathtt{x}.\mathtt{x}$ expresses that the definiens of `id` can be typed $\mathtt{a} \to \mathtt{a}$ for any `a`. A simple-minded but inefficient way to use `id` at many different types would be to replace each use of `id` with $\lambda \mathtt{x}.\mathtt{x}$ during type checking; with polymorphic types, we can instead determine a suitable instantiation of $\mathtt{a} \to \mathtt{a}$ at each use site of `id`.

We can likewise understand universe polymorphism as a mechanism for summarizing the possible *universe level assignments* of a top-level definition in dependent type theory. Rather than adding level quantification as a type former, we parametrize our judgments by an additional context of top-level definitions, the *signature* $\Sigma\ \mathrm{sig}_{\mathcal{H}}$ (Figure 2). Unlike the ordinary typing context $\Delta \vdash_{\mathcal{H}} \Gamma\ \mathrm{ctx}$ whose declarations $x_i{:}A_i$ all take place at the same level context Δ, every declaration $\varsigma_i{\sim}A_i@\Delta_i$ in Σ is annotated with its own level context Δ_i.[3] This allows each top-level definition to record its own set of level variables and level constraints; whenever a top-level declaration $\varsigma_i{\sim}A_i@\Delta_i$ is used by a term in level context Δ, it is under an explicit level substitution $\varsigma\{\sigma\}$ that instantiates Δ_i at Δ.

Remark. *Our signatures play a similar role to the principal schematic, generic definitional contexts of Harper and Pollack [1991], with the exception that we do not record the definiens of each ς_i. Although*

[3]The notation $\varsigma_i{\sim}A_i@\Delta_i$ is inspired by modal type theories where @ indicates the *mode* of a type; however, our development departs from modal type theory for reasons to be explained shortly.

the intended purpose of our signatures is to record top-level definitions, our formulation of universe polymorphism does not require access to definitions—indeed, as discussed above, it can be seen as a mechanism for avoiding *the need to expand definitions. (Conversion checking may require access to definitions due to type dependency, but this is orthogonal to universe-polymorphic type checking.)*

Remark. *One alternative to signatures would be to annotate ordinary variables in* Γ *with differing level contexts, in the style of multimodal type theory [Gratzer et al. 2020] but with level contexts playing the role of modes; thus* Γ *would subsume the need for signatures. Our formulation more closely matches what is used in practice; in particular, we do not wish to consider* Π*-types whose domains and codomains live at different universe hierarchies. Furthermore, we want explicit substitutions on types* $A\{\sigma\}$ *to judgmentally behave as ordinary substitutions* $A[\sigma]$ *(e.g.* $(\mathfrak{U}_i)\{i \mapsto 0\} \equiv \mathfrak{U}_0$*), which in multimodal type theory would correspond to a modality that commutes strictly with every type former. Such equations disrupt the adjunction between modal types and modal structure in contexts.*

The judgments of $\mathcal{H}$-polymorphic type theory are as follows:

- $\Sigma\ \mathrm{sig}_{\mathcal{H}}$.
- $\Delta \vdash^{\Sigma}_{\mathcal{H}} \Gamma\ \mathrm{ctx}$, presupposing $\Delta : \mathrm{SOrd}$ and $\Sigma\ \mathrm{sig}_{\mathcal{H}}$.
- $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\ell}$, presupposing $\Delta : \mathrm{SOrd}$, $\Sigma\ \mathrm{sig}_{\mathcal{H}}$, $\Delta \vdash^{\Sigma}_{\mathcal{H}} \Gamma\ \mathrm{ctx}$, and $\ell \in \mathcal{H}(\Delta)^{\top}$.
- $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A \equiv B\ \mathrm{type}_{\ell}$, presupposing as above, $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\ell}$, and $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} B\ \mathrm{type}_{\ell}$.
- $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} e : A$, presupposing $\Delta : \mathrm{SOrd}$, $\Sigma\ \mathrm{sig}_{\mathcal{H}}$, $\Delta \vdash^{\Sigma}_{\mathcal{H}} \Gamma\ \mathrm{ctx}$, and $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\top}$.
- $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} e_1 \equiv e_2 : A$, presupposing as above, $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} e_1 : A$, and $\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} e_2 : A$.

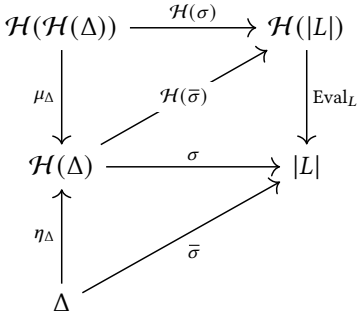
The bulk of the rules for $\mathcal{H}$-polymorphic type theory are identical to the rules for $\mathcal{H}(\Delta)$-monomorphic type theory (Figure 1) where Δ is the current level context. For instance:

$$\frac{\ell' <_{\mathcal{H}(\Delta)^{\top}} \ell}{\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} \mathfrak{U}^{\ell}_{\ell'}\ \mathrm{type}_{\ell}} \qquad \frac{\ell' \leq_{\mathcal{H}(\Delta)^{\top}} \ell \qquad \Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\ell'}}{\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} \Uparrow^{\ell}_{\ell'}(A)\ \mathrm{type}_{\ell}}$$

Judgments at different level contexts Δ, Ξ are related by level substitutions σ, which are morphisms between the *free* $\mathcal{H}$*-algebras* $F^{\mathcal{H}}(\Delta), F^{\mathcal{H}}(\Xi)$ on those contexts:

$$\frac{\Delta; \Gamma \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\ell} \qquad \sigma \in \mathrm{SOrd}^{\mathcal{H}}(F^{\mathcal{H}}(\Delta), F^{\mathcal{H}}(\Xi))}{\Xi; \Gamma[\sigma] \vdash^{\Sigma}_{\mathcal{H}} A[\sigma]\ \mathrm{type}_{\sigma(\ell)}}$$

Put simply, such level substitutions $\sigma \in \mathrm{SOrd}^{\mathcal{H}}(F^{\mathcal{H}}(\Delta), F^{\mathcal{H}}(\Xi))$ send level variables of Δ to level expressions in $\mathcal{H}(\Xi)$. This is because algebra morphisms out of free monad algebras are determined by their behavior on generators: from any $\mathcal{H}$-algebra morphism $\sigma : F^{\mathcal{H}}(\Delta) \to L$, we can construct a map $\overline{\sigma} : \Delta \to |L| = \sigma \circ \eta_{\Delta}$. Conversely, given $\overline{\sigma} : \Delta \to |L|$, we can construct an $\mathcal{H}$-algebra morphism $\sigma = \mathrm{Eval}_L \circ \mathcal{H}(\overline{\sigma})$. We thus have an isomorphism between $\mathcal{H}$-algebra morphisms $F^{\mathcal{H}}(\Delta) \to L$ and SOrd-morphisms $\Delta \to |L|$, as summarized in the diagram below:



$$\begin{array}{rl} \text{Signatures} & \Sigma \coloneqq \cdot \mid \Sigma, \varsigma{\sim}A@\Delta \\ \text{Terms} & e \coloneqq \ldots \mid \varsigma\{\sigma\} \end{array}$$

$$\frac{}{\cdot\ \mathrm{sig}_{\mathcal{H}}} \qquad \frac{\Sigma\ \mathrm{sig}_{\mathcal{H}} \qquad \Delta';\cdot \vdash^{\Sigma}_{\mathcal{H}} A\ \mathrm{type}_{\top} \qquad \varsigma \notin \Sigma}{\Sigma, \varsigma{\sim}A@\Delta'\ \mathrm{sig}_{\mathcal{H}}} \qquad \frac{\Sigma(\varsigma) = A@\Delta' \qquad \sigma \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta'), F(\Delta))}{\Delta;\Gamma \vdash^{\Sigma}_{\mathcal{H}} \varsigma\{\sigma\} : A[\sigma]}$$

$$(\varsigma\{\sigma\})[\delta] \coloneqq \varsigma\{\delta \circ \sigma\}$$

Fig. 2. Rules for signatures (top-level definitions) in $\mathcal{H}$-polymorphic type theory.

The remaining rules of $\mathcal{H}$-polymorphic type theory (in Figure 2) govern signatures: as we have already discussed, signatures Σ are contexts in which the type of each variable ς is at a different level context Δ, and therefore the "variable rule" for ς builds in an explicit level substitution σ.

Remark. *Mathematically, much of our theory remains sensible if we consider non-free $\mathcal{H}$-algebras, corresponding to level substitutions defined to map* non-variable *expressions to other expressions, such as $\alpha + 2 \mapsto \beta + 42$. In addition to being semantically unusual, we believe such substitutions would be (in general) inefficient in practice. On the other hand, such a generalization allows us to consider level contexts with strict inequalities between arbitrary level expressions, which seems potentially useful.*

Example 2.14 (Traditional level polymorphism). A simple universe-polymorphic type theory, with level variables $\alpha, \beta, \ldots$, concrete universes indexed by $\mathbb{N}$, and no other level expressions, is $\mathcal{H}$-polymorphic type theory for the `Either` monad $\mathcal{H} = \mathtt{Either}\ \mathbb{N}$. The intuition is that a level is either a variable or a concrete level ("error code"), and level substitutions (monadic computations) can send variables to either variables or concrete levels, but can never affect concrete levels. Writing $\iota_i : L_i \to L_0 \amalg L_1$ for the ith coproduct injection:

$$\begin{array}{rlrl} \mathcal{H}(\Delta) &\coloneqq \Delta \amalg \mathbb{N} & \eta_\Delta(\alpha) &\coloneqq \iota_0(\alpha) \\ \mathcal{H}(f)(\iota_0(\alpha)) &\coloneqq \iota_0(f(\alpha)) & \mu_\Delta(\iota_0(\ell)) &\coloneqq \ell \\ \mathcal{H}(f)(\iota_1(n)) &\coloneqq \iota_1(n) & \mu_\Delta(\iota_1(n)) &\coloneqq \iota_1(n) \end{array}$$

The fragment in which $\Delta = \emptyset$ is the $\mathbb{N}$-universe-monomorphic type theory of Example 2.2.

Example 2.15 (Semilattice of levels). Universe levels in Lean and Agda are populated by level variables, `zero`, `suc`$(-)$, and $- \vee -$ (maximum) operations satisfying a handful of laws: `zero` and $\vee$ form a bounded join semilattice, `suc` distributes over $\vee$, and $\alpha \vee \mathtt{suc}(\alpha) = \mathtt{suc}(\alpha)$.

These proof assistants do not directly fit into our framework for two reasons. First, unlike us, Agda *internalizes* the type of universe levels; however, our signatures of top-level definitions capture the essence of internal prenex level quantification, in that each definition generalizes over (and can be instantiated with) a collection of universe levels.[4]

A more serious issue is that accurately capturing the behavior of $\vee$ in these systems requires us to model universe lifting $\mathfrak{U}_i \to \mathfrak{U}_j$ and universe membership $\mathfrak{U}_i : \mathfrak{U}_j$ via two *different* relations, rather than as the non-strict $\leq$ and strict $<$ orders associated to a *single* relation as we have done above. To see why, observe that we want $\alpha \leq \alpha \vee \beta$ to ensure that we can lift $\Uparrow_\alpha^{\alpha\vee\beta} : \mathfrak{U}_\alpha \to \mathfrak{U}_{\alpha\vee\beta}$. But neither a strict inequality ($\alpha < \alpha \vee \beta$) nor an equality ($\alpha = \alpha \vee \beta$) gives us the correct notion of universe membership: the former implies $\mathfrak{U}_\alpha : \mathfrak{U}_{\alpha\vee\beta}$, which under $\beta \mapsto \alpha$ yields the inconsistent $\mathfrak{U}_\alpha : \mathfrak{U}_\alpha$ by $\alpha \vee \alpha = \alpha$, whereas the latter under $\beta \mapsto \mathtt{suc}(\alpha)$ implies $\alpha = \mathtt{suc}(\alpha)$ by $\alpha \vee \mathtt{suc}(\alpha) = \mathtt{suc}(\alpha)$.

[4]Our framework does *not* capture Agda's non-prenex level quantification.

In Section 5.1 we discuss a modification to our framework that decouples the non-strict/lifting and strict/membership relations, but for the moment we can instead approximate these systems by restricting what lifts exist. Setting aside `zero` and `suc` for simplicity, we can define $\mathcal{H}(\Delta)$ to add "formal joins" to Δ as follows:

Given a finite subset Ξ of a poset Δ, define the "tip" elements of Ξ as the set

$$\mathrm{Tips}_\Delta(\Xi) \coloneqq \{\alpha \mid \alpha \in \Xi \text{ and } \neg\exists\beta.(\alpha <_\Delta \beta)\}$$

and define the poset $\mathfrak{P}^+(\Delta)$ of finite subsets of Δ containing only tip elements as follows:

$$|\mathfrak{P}^+(\Delta)| \coloneqq \text{non-empty finite subsets of } \Delta \text{ of incomparable elements}$$
$$\Xi <_{\mathfrak{P}^+(\Delta)} \Theta \iff \forall\alpha{\in}\Xi, \exists\beta{\in}\Theta, \alpha <_\Delta \beta$$

The elements of $\mathfrak{P}^+(\Delta)$ represent non-empty formal joins of elements of Δ, subject to the limitation described above: concretely, the formal join of Ξ may not be $<_{\mathfrak{P}^+(\Delta)}$ the formal join of Θ even if $\Xi \subset \Theta$. (We exclude the empty set because it would be $<$ itself.)

Using these formal joins, we define a universe hierarchy monad as follows:

$$\begin{aligned}\mathcal{H}(\Delta) &\coloneqq \mathfrak{P}^+(\Delta)\\ \mathcal{H}(f)(\Xi) &\coloneqq \mathrm{Tips}(\{f(\alpha) \mid \alpha \in \Xi\})\\ \eta_\Delta(\alpha) &\coloneqq \{\alpha\}\\ \mu_\Delta(P) &\coloneqq \mathrm{Tips}(\textstyle\bigcup_{\Xi\in P}\Xi)\end{aligned}$$

We sketch the proof that $\mathcal{H}$ is a monad:

- $\mathcal{H}(f)$ preserves the strict order $<_{\mathcal{H}(\Delta)}$: Suppose $\Xi < \Theta$ and f is $<$-preserving; it suffices to show $\mathcal{H}(f)(\Xi) < \mathcal{H}(f)(\Theta)$. Choose any $\alpha' \in \mathcal{H}(f)(\Xi)$, and show there exists $\beta' \in \mathcal{H}(f)(\Theta)$ such that $\alpha' < \beta'$. For any $\alpha' \in \mathcal{H}(f)(\Xi)$, there exists $\alpha \in \Xi$ such that $\alpha' = f(\alpha)$. Then, because $\Xi < \Theta$, there exists $\beta \in \Theta$ such that $\alpha < \beta$. Therefore, there exists $\beta'' \in \mathcal{H}(f)(\Theta)$ such that $f(\beta) \le \beta''$, and $\alpha' = f(\alpha) < f(\beta) \le \beta''$ because f preserves the order $\alpha < \beta$.
- $\mathcal{H}(\mathbf{id})$ is the identity, and $\mathcal{H}(-)$ preserves composition.
- η_Δ preserves the strict order: if $\alpha < \beta \in \Delta$, then $\{\alpha\} < \{\beta\}$.
- μ_Δ preserves the strict order: if $P < Q \in \mathfrak{P}^+(\mathfrak{P}^+(\Delta))$, for any $\alpha \in \mu_\Delta(P)$, there exists $\Xi \in P$ such that $\alpha \in \Xi$. Because $P < Q$, there exists $\Theta \in Q$ such that $\Xi < \Theta$. Therefore, there exists $\beta \in \Theta$ such that $\alpha < \beta$, and $\beta' \in \mu_\Delta(Q)$ such that $\alpha < \beta \le \beta'$.

In Section 3 we will discuss our primary examples of universe-polymorphic type theories, those arising from McBride-style stratification.

2.3 Consistency of Generalized Universe Hierarchies

The universe-monomorphic and -polymorphic theories presented in this section include some rather unusual universe hierarchies as instances, e.g. infinite descending chains of universes (Example 2.3). One may therefore reasonably wonder whether these systems are consistent: type theorists typically establish consistency via some kind of model construction which uses the well-foundedness of the universe hierarchy as the basis of an induction order [Kovács 2022].

In this section we prove that any $\mathcal{H}$-polymorphic type theory is as consistent as standard (monomorphic) type theory with $\mathbb{N}$-indexed universes, essentially because any given proof can only mention finitely many universes, allowing us to interpret those universe levels as natural numbers while preserving the strict order. We therefore argue that type theorists should break the longstanding tradition of equating syntactic universe levels with the induction order used in building semantic models: the former need not even be well-founded (as in Examples 2.3 and 2.4). The key to consistent syntax is simply irreflexivity, that is, preventing $\mathfrak{U} : \mathfrak{U}$.

Lemma 2.16. *Let L be a poset. For any derivation $\mathfrak{D}$ in L-monomorphic type theory, there is a finite poset L', a full monomorphism $\sigma \in \mathrm{SOrd}(L', L)$, and a derivation $\mathfrak{D}'$ in L'-monomorphic type theory such that $\mathfrak{D}'[\sigma] = \mathfrak{D}$.*

Proof. Let L' be the subobject of L containing those universe levels that index type judgments in $\mathfrak{D}$, and let σ be the inclusion $L' \to L$. The poset L' is finite because inference rules in L-monomorphic type theory mention only finite numbers of universe levels, and derivations consist of finitely many inference rules. We obtain the L'-derivation $\mathfrak{D}'$ by replacing all levels $\ell \in L$ in $\mathfrak{D}$ with their preimages in L'. By construction, $\mathfrak{D}'[\sigma] = \mathfrak{D}$. □

We can then relate the consistency of $\mathcal{H}$-polymorphic type theory to that of $\mathbb{N}$-monomorphic type theory in a series of steps:

Lemma 2.17 (Consistency of universe-monomorphic type theory). *For any poset L, if L-monomorphic type theory is inconsistent, then $\mathbb{N}$-monomorphic type theory is inconsistent.*

Proof. Suppose there is a derivation $\mathfrak{D}$ of $\cdot \vdash_L e : \mathbf{0}^\top$ in L-monomorphic type theory. By Lemma 2.16, there is a derivation of $\cdot \vdash_{L'} e' : \mathbf{0}^\top$ in L'-monomorphic type theory for some finite L' and some term e'. But any finite $L' : \mathrm{SOrd}$ admits a morphism $L' \to \mathbb{N}$, and thus by Lemma 2.7, there is a derivation of $\cdot \vdash_{\mathbb{N}} e'' : \mathbf{0}^\top$ in $\mathbb{N}$-monomorphic type theory for some term e'', which is to say that $\mathbb{N}$-monomorphic type theory is inconsistent. □

Lemma 2.18. *For any hierarchy theory $\mathcal{H}$, the judgment $\Delta; \Gamma \vdots_{\mathcal{H}} e : A$ is derivable in $\mathcal{H}$-polymorphic type theory if and only if $\Gamma \vdash_{\mathcal{H}(\Delta)} e : A$ is derivable in $\mathcal{H}(\Delta)$-monomorphic type theory.*

Proof. By induction on the derivation. □

Theorem 2.19 (Consistency of universe-polymorphic type theory). *For any hierarchy theory $\mathcal{H}$, if $\mathcal{H}$-polymorphic type theory is inconsistent, then $\mathbb{N}$-monomorphic type theory is inconsistent.*

Proof. Suppose the judgment $\Delta; \cdot \vdots_{\mathcal{H}} e : \mathbf{0}^\top$ is derivable for some poset Δ and some term e in $\mathcal{H}$-polymorphic type theory. By Lemma 2.18, $\mathcal{H}(\Delta)$-monomorphic type theory is inconsistent, and thus by Lemma 2.17, $\mathbb{N}$-monomorphic type theory is inconsistent. □

3 CRUDE BUT UNIVERSAL STRATIFICATION

McBride has proposed *displacement* as a "crude but effective" mechanism for universe polymorphism, in which top-level definitions are written in a monomorphic fashion but can be uniformly lifted $(-)^{\Uparrow}$ at their use sites to higher universe levels [McBride 2002, 2011]. For example, a user might define the (type-)polymorphic identity function at $\mathfrak{U}_0$:

$$\mathtt{id} : \textstyle\prod_{A:\mathfrak{U}_0} (A \to A)$$

To apply this function to a type in $\mathfrak{U}_n$, we can lift, or displace, the entire definition by n universe levels, yielding:

$$\mathtt{id}^{\Uparrow n} : \textstyle\prod_{A:\mathfrak{U}_n} (A \to A)$$

Thus one writes $\mathtt{id}^{\Uparrow 1}(\mathfrak{U}_0)(A)$ to apply the identity function to $A : \mathfrak{U}_0$, using the fact that $\mathfrak{U}_0 : \mathfrak{U}_1$.

Rather than considering displacement $\Uparrow^1$ to "substitute 1 for 0," we can understand McBride's scheme as implicitly indexing each top-level definition by a single universe level α, where

$$\alpha; \cdot \vdash \mathtt{id}^{\alpha} : \textstyle\prod_{A:\mathfrak{U}_\alpha} (A \to A)$$

Then displacement $(-)^{\Uparrow n}$ performs the level substitution $[\alpha \mapsto \alpha + n]$, and uses of top-level definitions implicitly instantiate $[\alpha \mapsto 0]$. More abstractly, one might imagine universe levels as pairs of a level variable α with a natural number n denoting the total displacement of the variable α; then $(-)^{\Uparrow n}$ sends the universe level (α, m) to $(\alpha, m + n)$.

Remark. *The more abstract reformulation above can be understood through the lens of Cayley's theorem, which establishes an isomorphism between a group G and the multiplication action of G on itself (as a subgroup of the automorphism group of $|G|$), in this case regarding $n \in \mathbb{N}$ as* addition by n *$\lambda\alpha.\alpha + n$. This functional interpretation is a useful conceptual leap toward Lemma 3.8.*

In the remainder of this section, we generalize McBride's displacement operation, then prove that any hierarchy theory $\mathcal{H}$ can be understood as an instance of this generalized displacement.

3.1 Algebras of Displacements

We will start by rephrasing $\mathbb{N}$-displacement as a hierarchy theory (a monad on SOrd); then, we will investigate what is the minimum algebraic structure needed to perform displacement for a non-$\mathbb{N}$ poset. Our displacement monad is defined in terms of the following construction:

Definition 3.1 (Left-invariant right-centered product). Given posets A and B and an element $b \in B$, their left-invariant right-centered product $A \ltimes_b B$ is the following poset (defined via its *strict* order):

$$\begin{aligned} &|A \ltimes_b B| \coloneqq |A| \times |B| \\ &(a_1, b_1) <_{A \ltimes_b B} (a_2, b_2) \iff \\ &\qquad (a_1 = a_2 \text{ and } b_1 <_B b_2) \text{ or } (a_1 <_A a_2 \text{ and } b_1 \leq_B b \leq_B b_2) \end{aligned}$$

Remark. *The strict order $<_{A \ltimes_b B}$ is the least relation generated by:*

(1) *$(a, b_1) < (a, b_2)$ for any $a \in A$ and any $b_1 <_B b_2 \in B$.*
(2) *$(a_1, b) < (a_2, b)$ for any $a_1 <_A a_2 \in A$ when $b \in B$ is the given element (the center of B).*

Condition (1) expresses the constraint that the level $\alpha + 10$ should be greater than $\alpha + 5$. Condition (2) states that displacement should respect the strict order of A when at the center *of B; this is needed to define the unit η of the hierarchy theory. The center is intended to represent the identity displacement; the reason why we restrict condition (2) to the center of B (rather than arbitrary elements of B) is that, later in Section 3.4, a critical construction used in the proof of Lemma 3.8 will not necessarily preserve the strict order (2) for non-center elements of B. (See the remark on Page 17.)*

Using Definition 3.1, we can express $\mathbb{N}$-displacement as the following "McBride monad" $\mathcal{M}$:

$$\begin{aligned} \mathcal{M}(\Delta) &\coloneqq \Delta \ltimes_0 \mathbb{N} \\ \mathcal{M}(\sigma)(\alpha, n) &\coloneqq (\sigma(\alpha), n) \qquad \text{(functoriality)} \\ \eta_\Delta(\alpha) &\coloneqq (\alpha, 0) \\ \mu_\Delta((\alpha, n_1), n_2) &\coloneqq (\alpha, n_1 + n_2) \end{aligned}$$

The monad $\mathcal{M}$ already generalizes McBride's "crude but effective stratification" from operating only on definitions parameterized by a single level variable α, to terms parameterized by arbitrary level contexts Δ. What about generalizing $\mathbb{N}$? The necessary components are a set of displacements $\mathcal{D}$ with a partial order $\leq_\mathcal{D}$, a binary operator $\bullet$ on $\mathcal{D}$, and an element $\star \in \mathcal{D}$ (likely a unit). Thus, suppose that instead of $(\mathbb{N}, +, 0, \leq)$ we have an arbitrary pointed magma with a partial order $(\mathcal{D}, \bullet, \star, \leq_\mathcal{D})$, and we build the following candidate hierarchy theory $\mathcal{M}_\mathcal{D}$:

$$\begin{aligned} \mathcal{M}_\mathcal{D}(\Delta) &\coloneqq \Delta \ltimes_\star \mathcal{D} \\ \mathcal{M}_\mathcal{D}(f)(\alpha, d) &\coloneqq (f(\alpha), d) \\ \eta_\Delta(\alpha) &\coloneqq (\alpha, \star) \\ \mu_\Delta((\alpha, d_1), d_2) &\coloneqq (\alpha, d_1 \bullet d_2) \end{aligned}$$

Under what conditions on $(\mathcal{D}, \bullet, \star, \leq_\mathcal{D})$ is $\mathcal{M}_\mathcal{D}$ a hierarchy theory?

- $\mathcal{M}_{\mathcal{D}}$ must be a functor SOrd $\to$ SOrd, i.e. for any $<$-preserving map $f : \Delta_1 \to \Delta_2$,
$$\mathcal{M}_{\mathcal{D}}(f)(\alpha, d) \coloneqq (f(\alpha), d)$$
should preserve the strict order of $\Delta_1 \ltimes_\star \mathcal{D}$. Suppose $(\alpha, d) < (\beta, e)$; we need $(f(\alpha), d) < (f(\beta), e)$. Unfolding the definition of $\ltimes_\star$, there are two cases to consider:
(1) $\alpha = \beta$ and $d <_{\mathcal{D}} e$. The condition already holds by the functoriality of f.
(2) $\alpha < \beta$ and $d \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e$. The condition already holds because f is $<$-preserving.
We also have to check that $\mathcal{M}_{\mathcal{D}}(-)$ preserves identity and composition, which is immediate.
- Functions η_Δ already preserve the strict order by the definition of $\ltimes_\star$. For any $\alpha_1 <_\Delta \alpha_2$, we have $(\alpha_1, \star) <_{\mathcal{D}} (\alpha_2, \star) \in \Delta \ltimes_\star \mathcal{D}$ under no additional hypotheses.
- Functions μ_Δ must also preserve the strict order. Suppose $((\alpha, d_1), d_2) < ((\beta, e_1), e_2)$; we need $(\alpha, d_1 \bullet d_2) < (\beta, e_1 \bullet e_2)$. By the definition of $\ltimes_\star$, there are three cases:
(1) $\alpha = \beta$, $d_1 = e_1$, and $d_2 <_{\mathcal{D}} e_2$. The condition holds if $(d_1 \bullet d_2) <_{\mathcal{D}} (e_1 \bullet e_2)$; that is, if the order $<_{\mathcal{D}}$ is left-invariant with respect to the operator $\bullet$.
(2) $\alpha = \beta$, $d_1 <_{\mathcal{D}} e_1$, and $d_2 \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e_2$. Thanks to transitivity, the condition holds if $\star$ is a right unit of $\bullet$ and the non-strict order $\leq_{\mathcal{D}}$ is left-invariant with respect to the operator $\bullet$ (a condition weaker than the strict order $<_{\mathcal{D}}$ being left-invariant):
$$(d_1 \bullet d_2) \leq_{\mathcal{D}} (d_1 \bullet \star) = d_1 <_{\mathcal{D}} e_1 = (e_1 \bullet \star) \leq_{\mathcal{D}} (e_1 \bullet e_2)$$
(3) $\alpha < \beta$, $d_1 \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e_1$, and $d_2 \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e_2$. The condition holds if $(d_1 \bullet d_2) \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} (e_1 \bullet e_2)$, which in turn holds if $\star$ is a right unit and $\leq_{\mathcal{D}}$ is left-invariant:
$$(d_1 \bullet d_2) \leq_{\mathcal{D}} (d_1 \bullet \star) = d_1 \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e_1 = (e_1 \bullet \star) \leq_{\mathcal{D}} (e_1 \bullet e_2)$$
or alternatively if $\star$ is a left unit and $\leq_{\mathcal{D}}$ is right-invariant:
$$(d_1 \bullet d_2) \leq_{\mathcal{D}} (\star \bullet d_2) = d_2 \leq_{\mathcal{D}} \star \leq_{\mathcal{D}} e_2 = (\star \bullet e_2) \leq_{\mathcal{D}} (e_1 \bullet e_2)$$
- η and μ are natural transformations immediately by definition.
- The monad laws require $\mu \circ \mathcal{M}(\mu) = \mu \circ \mu_{\mathcal{M}(-)}$ and $\mu \circ \mathcal{M}(\eta) = \mu \circ \eta_{\mathcal{M}(-)} = \mathbf{id}$. Unfolding definitions, these assert that $\bullet$ is associative and $\star$ is a (left and right) unit of $\bullet$.

In sum, $\mathcal{M}_{\mathcal{D}}$ is a monad when $(\mathcal{D}, \bullet, \star, \leq_{\mathcal{D}})$ satisfies the following properties:

(1) The binary operator $\bullet$ is associative.
(2) The element $\star$ is a unit of the operator $\bullet$.
(3) The strict order $<_{\mathcal{D}}$ is left-invariant with respect to the operator $\bullet$.

We have mechanized the above statement in Agda. Magmas satisfying the non-strict version of condition (3) are known as *left-order magmas* [Ha and Harizanov 2018], and thus the algebras we consider might reasonably be called *strictly-left-order monoids*. (They are almost ordered monoids, except that the weak order is not necessarily right-invariant with respect to the binary operator.) Due to the central role of these algebras in our work, we will simply call them *displacement algebras*.

Definition 3.2 (Displacement algebra). A *displacement algebra* is a monoid with a partial order $(\mathcal{D}, \bullet, \star, \leq_{\mathcal{D}})$ such that the strict order $<_{\mathcal{D}}$ is left-invariant:
$$d_1 <_{\mathcal{D}} d_2 \implies (d \bullet d_1) <_{\mathcal{D}} (d \bullet d_2) \text{ for any } d \in \mathcal{D}$$

McBride's original displacement operation corresponds to the displacement algebra $(\mathbb{N}, +, 0, \leq)$. In Section 3.4, we will argue that displacement algebras capture the essence of hierarchy theories, including ones that are not obviously instances of $\mathcal{M}_{\mathcal{D}}$ for some $\mathcal{D}$.

Remark. *McBride [2002] was already aware that any class of strictly monotone displacements with identity and composition suffices for "crude but effective stratification." Our contribution is a much*

more careful analysis of his idea within our framework of universe hierarchy monads: we show above that these conditions are not only sufficient but also necessary *for appropriate universe polymorphism.*

3.2 Augmented Displacement Algebras

We have just seen that McBride's "crude but effective stratification" can be implemented using any displacement algebra $(\mathcal{D}, \bullet, \star, \leq)$. In practice, however, it may be useful to place several additional conditions on displacement algebras:

Right-invariance of $\leq$. One might expect that substituting a larger universe level into a level expression does not make the resulting expression smaller. This condition is not needed for the correctness of the type theory, but may be more intuitive to users. A displacement algebra $\mathcal{D}$ enjoys right-invariance of $\leq$ if and only if $(\mathcal{D}, \leq)$ is also an ordered monoid.

Joins. A join operator on displacements is useful for levels involving multiple variables. Note that this is not the join operator on whole level expressions, but only on displacements.

Bottom. A bottom element $\bot$ corresponds to a minimum displacement. Note that this might not be the unit of $\bullet$, though in McBride's setting $0 \in \mathbb{N}$ plays both roles. Having $\bot$ allows us to define a join operator on lexicographically ordered tuples, setting the join of the pairs (d_1, e_1) and (d_2, e_2) to be $(d_1 \vee d_2, \bot)$ when the join of the first components, $(d_1 \vee d_2)$, is strictly greater than both d_1 and d_2. We have not found $\bot$ useful in other situations.

Constants. $\mathbb{N}$-displacement has displacement *by* a constant ($\lambda\alpha.\alpha + 10$), but not displacement *to a constant level* ($\lambda\alpha.10$). We do not believe these truly constant displacements are useful; that said, in Section 3.3 we provide a general construction of truly constant displacements.

Successors. A successor operator gives the minimal level $\ell + 1$ containing the type $\mathfrak{U}_\ell$, which is helpful for type inference. However, many of our examples do not admit successors.

In Section 3.3 we will consider examples with a join operator, a bottom element, or truly constant displacements. We adopt the following terminology:

Definition 3.3 (Displacement semilattice). A *displacement semilattice* $\mathcal{D}$ is a displacement algebra equipped with a join operator "$\vee$" with respect to its non-strict order.

3.3 Examples of Displacement Algebras

We now present a handful of examples of displacement algebras, starting with more familiar ones. All the examples have been mechanized in Agda and shown to satisfy the axioms of displacement algebras (Definition 3.2).

3.3.1 Natural Numbers, Integers, and Non-Positive Integers. $(\mathbb{N}, +, 0, \leq)$ along with 0 and max is a bounded displacement semilattice. This arguably matches the conventional universe levels the best. $(\mathbb{Z}, +, 0, \leq)$ along with max is a displacement semilattice, and both its non-negative ($\mathbb{N}$) and non-positive fragments are its displacement subsemilattices.

3.3.2 Constant Displacements.

Definition 3.4. A *right displacement-action* of a displacement algebra $\mathcal{D}$ on a poset S is a set function $f : |S| \times |\mathcal{D}| \to |S|$ that:

Respects unit: $f(s)(\star) = s$;
Respects composition: $f(s)(d \bullet d') = f(f(s)(d))(d')$; and
Respects the strict order of $\mathcal{D}$: $d <_{\mathcal{D}} d' \implies f(s)(d) <_S f(s)(d')$.

Suppose we have a displacement algebra $\mathcal{D}$, a poset C ("constants"), and a right $\mathcal{D}$-action f on C. We may construct a displacement algebra $\mathcal{D}^{\text{const}} = (|\mathcal{D}^{\text{const}}|, \bullet, \star, \leq)$ as follows:

$$|\mathcal{D}^{\text{const}}| := |\mathcal{D}| \amalg |C|$$

$$
\begin{aligned}
\_ \bullet \iota_1(c) &\coloneqq \iota_1(c) \\
\iota_0(d) \bullet \iota_0(d') &\coloneqq \iota_0(d \bullet_{\mathcal{D}} d') \\
\iota_1(c) \bullet \iota_0(d) &\coloneqq \iota_1(f(c)(d)) \\
\star &\coloneqq \iota_0(\star_{\mathcal{D}}) \\
d \leq d' &\iff (d = \iota_0(\hat{d}) \text{ and } d' = \iota_0(\hat{d}') \text{ and } \hat{d} \leq_{\mathcal{D}} \hat{d}') \\
&\qquad \text{or } (d = \iota_1(\hat{c}) \text{ and } d' = \iota_1(\hat{c}') \text{ and } \hat{c} \leq_{C} \hat{c}')
\end{aligned}
$$

Definition 3.5. A right displacement-action of $\mathcal{D}$ on S is *weakly right-invariant* if $a \leq b$ implies that $f(a)(x) \leq f(b)(x)$ for every $a, b : |S|$ and $x : |\mathcal{D}|$.

If $\mathcal{D}$ is an ordered monoid and f is weakly right-invariant, then $\mathcal{D}^{\text{const}}$ is also an ordered monoid.

3.3.3 Binary Products. Suppose we have two displacement algebras $\mathcal{D}_1$ and $\mathcal{D}_2$. We may construct a new displacement algebra $\mathcal{D} = (|\mathcal{D}|, \bullet, \star, \leq)$ as follows:

$$
\begin{aligned}
|\mathcal{D}| &\coloneqq |\mathcal{D}_1| \times |\mathcal{D}_2| \\
(d_1, d_2) \bullet (d_1', d_2') &\coloneqq (d_1 \bullet_{\mathcal{D}_1} d_1', d_2 \bullet_{\mathcal{D}_2} d_2') \\
\star &\coloneqq (\star_{\mathcal{D}_1}, \star_{\mathcal{D}_2}) \\
(d_1, d_2) \leq (d_1', d_2') &\iff d_1 \leq_{\mathcal{D}_1} d_1' \text{ and } d_2 \leq_{\mathcal{D}_1} d_2'
\end{aligned}
$$

If both $\mathcal{D}_1$ and $\mathcal{D}_2$ are ordered monoids, so is $\mathcal{D}$. If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have a bottom element, so does $\mathcal{D}$. If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have joins, so does $\mathcal{D}$.

3.3.4 Lexicographic Binary Products. Suppose we have two displacement algebras $\mathcal{D}_1$ and $\mathcal{D}_2$. We may construct a new displacement algebra $\mathcal{D} = (|\mathcal{D}|, \bullet, \star, \leq)$ as follows:

$$
\begin{aligned}
|\mathcal{D}| &\coloneqq |\mathcal{D}_1| \times |\mathcal{D}_2| \\
(d_1, d_2) \bullet (d_1', d_2') &\coloneqq (d_1 \bullet_{\mathcal{D}_1} d_1', d_2 \bullet_{\mathcal{D}_2} d_2') \\
\star &\coloneqq (\star_{\mathcal{D}_1}, \star_{\mathcal{D}_2}) \\
(d_1, d_2) \leq (d_1', d_2') &\iff d_1 <_{\mathcal{D}_1} d_1' \text{ or } (d_1 = d_1' \text{ and } d_2 \leq_{\mathcal{D}_2} d_2')
\end{aligned}
$$

If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have a bottom element, so does $\mathcal{D}$. If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have joins *and $\mathcal{D}_2$ has a bottom element*, $\mathcal{D}$ has joins as follows:

$$
(d_1, d_2) \vee (d_1', d_2') = \begin{cases}
(d_1 \vee d_1', \bot_{\mathcal{D}_2}) & d_1 < d_1 \vee d_1' \text{ and } d_1' < d_1 \vee d_1' \\
(d_1 \vee d_1', d_2) & d_1 = d_1 \vee d_1' \text{ and } d_1' < d_1 \vee d_1' \\
(d_1 \vee d_1', d_2') & d_1 < d_1 \vee d_1' \text{ and } d_1' = d_1 \vee d_1' \\
(d_1 \vee d_1', d_2 \vee d_2') & d_1 = d_1 \vee d_1' \text{ and } d_1' = d_1 \vee d_1'
\end{cases}
$$

3.3.5 Infinite Products. Suppose we have a displacement algebra $\mathcal{D}$. We may construct a new displacement algebra $\mathcal{D}^{\infty} = (|\mathcal{D}^{\infty}|, \bullet, \star, \leq)$ as follows:

$$
\begin{aligned}
|\mathcal{D}^{\infty}| &\coloneqq \mathbb{N} \to |\mathcal{D}| \\
d \bullet d' &\coloneqq \lambda(i{:}\mathbb{N}).d(i) \bullet_{\mathcal{D}} d'(i) \\
\star &\coloneqq \lambda(i{:}\mathbb{N}).\star_{\mathcal{D}} \\
d \leq d' &\iff \forall i.d(i) \leq d'(i)
\end{aligned}
$$

If $\mathcal{D}$ is an ordered monoid, so is $\mathcal{D}^{\infty}$, and if $\mathcal{D}$ has a bottom element (resp., joins), so does $\mathcal{D}^{\infty}$. In practice, one can limit $d \in \mathcal{D}^{\infty}$ to functions which are nearly constant except for only finite entries (that is, there exists $d^* \in \mathcal{D}$ such that for only a finite number of i, $d(i) \neq d^*$); such

functions constitute a bounded displacement subsemilattice for which equality is decidable. One can further fix d^* to be $\star_{\mathcal{D}}$, i.e., restrict to functions with finite support, which form a displacement subsemilattice (that in general lacks a bottom element).

3.3.6 Prefix Displacements. Given any set S (with decidable equality, if in a constructive setting), we may construct a displacement algebra $\mathcal{D} = (|\mathcal{D}|, \bullet, \star, \leq)$ as follows:

$$\begin{aligned} |\mathcal{D}^{\mathrm{pre}}| &:= \mathrm{List}(S) \\ d \bullet d' &:= \mathrm{Concat}(d, d') \\ \star &:= \mathrm{Nil} \\ d \leq d' &\iff d \text{ is a prefix of } d' \end{aligned}$$

$\mathcal{D}$ has a bottom element Nil. This example is inspired by thread forking in concurrent programming, where the universe at level ℓ represents the types inherited from its parent (prefix) threads. Spawning a child thread corresponds to appending a fresh symbol to the end of the list, and the universe hierarchy prevents access to types in its siblings. We do not have a particular use case of prefix displacements in mind in the context of theorem provers, but displacement algebras are flexible enough to include them as instances. Note that the join operation is by design not definable, because the larger universes (descendant threads) diverge.

3.3.7 Fractal Displacements. Suppose we have a displacement algebra $\mathcal{D}$. We may consider its lexicographically ordered infinite products as a new displacement algebra $\mathcal{D}^{\mathrm{frac}} = (|\mathcal{D}^{\mathrm{frac}}|, \bullet, \star, \leq)$:

$$\begin{aligned} |\mathcal{D}^{\mathrm{frac}}| &:= \mathrm{NonEmptyList}(|\mathcal{D}|) \\ \mathrm{Cons}(d_1, \mathrm{Nil}) \bullet \mathrm{Cons}(d_1', d_2') &:= \mathrm{Cons}(d_1 \bullet_{\mathcal{D}} d_1', d_2') \\ \mathrm{Cons}(d_1, \mathrm{Cons}(d_2, d_3)) \bullet d' &:= \mathrm{Cons}(d_1, \mathrm{Cons}(d_2, d_3) \bullet d') \\ \star &:= \mathrm{Cons}(\star_{\mathcal{D}}, \mathrm{Nil}) \end{aligned}$$

$$\begin{aligned} \mathrm{Cons}(d_1, \mathrm{Nil}) \leq \mathrm{Cons}(d_1', -) &\iff d_1 \leq_{\mathcal{D}} d_1' \\ \mathrm{Cons}(d_1, \mathrm{Cons}(-, -)) \leq \mathrm{Cons}(d_1, \mathrm{Nil}) &\iff d_1 <_{\mathcal{D}} d_1' \\ \mathrm{Cons}(d_1, \mathrm{Cons}(d_2, d_3)) \leq \mathrm{Cons}(d_1', \mathrm{Cons}(d_2', d_3')) &\iff \\ d_1 <_{\mathcal{D}} d_1' \text{ or } (d_1 = d_1' \text{ and } \mathrm{Cons}(d_2, d_3) &\leq \mathrm{Cons}(d_2', d_3')) \end{aligned}$$

This algebra embeds the entire universe hierarchy as sublevels between any two universe levels. As an analogy, one can embed the real line into the segment between 0 and 1. Each displacement is a list of displacements at different magnitudes, from the most significant to the least.

3.3.8 Opposite Displacements. Suppose we have a displacement algebra $\mathcal{D}$. We may construct a new displacement algebra $\mathcal{D}^{\mathrm{op}} = (|\mathcal{D}^{\mathrm{op}}|, \bullet, \star, \leq)$ as follows:

$$\begin{aligned} |\mathcal{D}^{\mathrm{op}}| &:= |\mathcal{D}| \\ d \bullet d' &:= d \bullet_{\mathcal{D}} d' \\ \star &:= \star_{\mathcal{D}} \\ d \leq d' &\iff d \geq_{\mathcal{D}} d' \end{aligned}$$

If $\mathcal{D}$ is an ordered monoid, so is $\mathcal{D}^{\mathrm{op}}$. This example demonstrates that we can freely flip the order of displacements, making the larger universes become the smaller ones.

3.4 Universal Hierarchy Theory

We now prove that any hierarchy theory $\mathcal{H}$ (over a small collection of small level contexts) embeds into a hierarchy theory of the form $\mathcal{M}_{\mathcal{D}}$ (Theorem 3.10). Therefore, our generalized notion of "crude but effective" displacement is sufficiently expressive to capture all universe hierarchy theories. The main idea behind our construction is that we can *represent* (in the category-theoretic sense) any hierarchy theory by a collection of endomorphism monoids, verifying first that these endomorphism monoids form displacement algebras.

Definition 3.6. Let $\mathcal{H}$ be a hierarchy theory and Δ be a poset. We write $\mathcal{D}^{\mathcal{H}}_{\Delta}$ for $\mathrm{Endo}_{\mathrm{SOrd}^{\mathcal{H}}}(F^{\mathcal{H}}(\Delta))$, the endomorphism monoid of $F^{\mathcal{H}}(\Delta)$, equipped with the pointwise (non-strict) order:

$$
\begin{aligned}
|\mathcal{D}^{\mathcal{H}}_{\Delta}| &:= \mathrm{SOrd}^{\mathcal{H}}(F^{\mathcal{H}}(\Delta), F^{\mathcal{H}}(\Delta)) \\
\sigma \bullet \delta &:= \sigma \circ \delta \\
\star &:= \mathbf{1}_{F^{\mathcal{H}}(\Delta)} \\
\sigma \leq_{\mathcal{D}^{\mathcal{H}}_{\Delta}} \delta &\iff \forall \alpha \in \Delta, \sigma(\eta_{\Delta}(\alpha)) \leq_{\mathcal{H}(\Delta)} \delta(\eta_{\Delta}(\alpha))
\end{aligned}
$$

Lemma 3.7. *For any hierarchy theory $\mathcal{H}$ and poset Δ, $\mathcal{D}^{\mathcal{H}}_{\Delta}$ is a displacement algebra.*

Proof. The strict order of $\mathcal{D}^{\mathcal{H}}_{\Delta}$ is left-invariant because all morphisms preserve strict orders. □

Remark. *The strict order is* not *right-invariant! When $\sigma < \delta$, we know $\sigma(\eta_{\Delta}(\alpha)) < \delta(\eta_{\Delta}(\alpha))$ for some $\alpha \in \Delta$, but not $\sigma(\eta_{\Delta}(\alpha')) < \delta(\eta_{\Delta}(\alpha'))$ for a different $\alpha' \in \Delta$, not to mention $\sigma \bullet \rho < \delta \bullet \rho$ for an arbitrary $\rho \in \mathcal{D}^{\mathcal{H}}_{\Delta}$. This is why displacement algebras (Definition 3.2) only require left-invariance. One special case is $\rho = \star$, which does imply $\sigma \bullet \rho < \delta \bullet \rho$, and this case has led to right-centeredness in the left-invariant right-centered products (Definition 3.1). We need right-centeredness to make the McBride monad $\mathcal{M}_{\mathcal{D}^{\mathcal{H}}_{\Delta}}$ a monad, but not more than that because $\mathcal{D}^{\mathcal{H}}_{\Delta}$ might not support more equations. That is, to show that a construction based on a left-invariant right-centered product is a monad, a displacement algebra suffices, but a construction based on a product with general right-invariance will in turn require the underlying displacement algebra to be right-invariant with respect to its strict order.*

To claim that we can embed a hierarchy theory $\mathcal{H}$, we need to embed both the elements of $\mathcal{H}$-algebras and the morphisms between them. The embedding involves two steps:

(1) Under reasonable assumptions, there is a large enough free algebra $F^{\mathcal{H}}(\Delta)$ whose endomorphisms and elements can embed all morphisms and elements from small free algebras.
(2) Endomorphisms and elements of $F^{\mathcal{H}}(\Delta)$ can be embedded into those of $F^{\mathcal{M}_{\mathcal{D}}}(\Psi)$ for some $\mathcal{D}$ and Ψ, where $\mathcal{M}_{\mathcal{D}}$ is the "generalized McBride monad" for the displacement algebra $\mathcal{D}$.

We show the second step first because it is easier. Definition 3.6 takes care of the endomorphisms, but we have to embed elements of the $\mathcal{H}$-algebra, too. Thus, we consider $\{\diamond\} \amalg \Delta \amalg \Delta$ instead of just Δ to give us extra room to embed elements of the $\mathcal{H}$-algebra as endomorphisms. In the following lemma, the embedding of endomorphisms is formulated as a functor T and the embedding of elements is formulated as a natural transformation ν so that the square on the right commutes:

$$
\begin{array}{ccc}
F^{\mathcal{H}}(\Delta) & \xrightarrow{\alpha} & F^{\mathcal{H}}(\Delta) \\
F^{\mathcal{M}_{\mathcal{D}}}(\Psi) & \xrightarrow{T(\alpha)} & F^{\mathcal{M}_{\mathcal{D}}}(\Psi)
\end{array}
\qquad\qquad
\begin{array}{ccc}
\mathcal{H}(\Delta) & \xrightarrow{U(\alpha)} & \mathcal{H}(\Delta) \\
\downarrow{\scriptstyle \nu_{F(\Delta)}} & & \downarrow{\scriptstyle \nu_{F(\Delta)}} \\
\mathcal{M}_{\mathcal{D}}(\Psi) & \xrightarrow{U(T(\alpha))} & \mathcal{M}_{\mathcal{D}}(\Psi)
\end{array}
$$

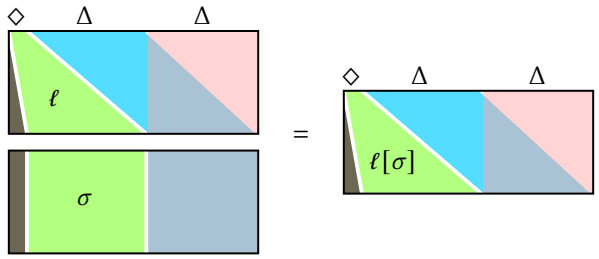


Fig. 3. A visualization of the "dodging" in the proof of Lemma 3.8.

Lemma 3.8. *Let $(\mathcal{H}, \eta, \mu)$ be any hierarchy theory, Δ be any poset, $\{\diamond\}$ be any singleton set, and Ψ be any non-empty set.*[5] *Let $\mathcal{D}$ be the displacement algebra $\mathcal{D}^{\mathcal{H}}_{\{\diamond\} \amalg \Delta \amalg \Delta}$. There is a functor T from the full subcategory of $\mathsf{SOrd}^{\mathcal{H}}$ on the object $F^{\mathcal{H}}(\Delta)$ to the full subcategory of $\mathsf{SOrd}^{\mathcal{M}_{\mathcal{D}}}$ on the object $F^{\mathcal{M}_{\mathcal{D}}}(\Psi)$. Moreover, there is a natural transformation $U^{\mathcal{H}} \to U^{\mathcal{M}_{\mathcal{D}}} \circ T$, and if $\mathcal{H}$ preserves monos, T is faithful.*

The idea behind $\{\diamond\} \amalg \Delta \amalg \Delta$ is that $\{\diamond\}$ is for embedding elements (not endomorphisms) of the free $\mathcal{H}$-algebra. The distinguished variable $\diamond$ can be mapped to those elements, using the variables in the first copy of Δ. The first copy of Δ is where subsequent substitutions happen and is where we should embed endomorphisms. The second copy of Δ is a buffer to prevent the original first copy of Δ from colliding with the variables in embedded elements: when an embedded element is using the first copy of Δ, the original first copy of Δ "dodges out" into the second copy of Δ. Such dodging is needed for the embedding to commute with the application of endomorphisms. Figure 3 is a visualization of the dodging: The left-hand side represents the composition of embedded ℓ and embedded σ, while the right-hand side represents the embedding of $\ell[\sigma]$. Both sides must agree. If the first copy of Δ (in blue) did not dodge out on the left-hand side, it would be further affected by the embedded σ, while the first copy of Δ on the right-hand side will never be affected by σ.

Proof. Let $\Delta^+ = \{\diamond\} \amalg \Delta \amalg \Delta$. T must send all objects to $F^{\mathcal{M}_{\mathcal{D}}}(\Psi)$. For morphisms, we will first embed $\sigma \in \mathsf{SOrd}^{\mathcal{H}}(F(\Delta), F(\Delta))$ into $\mathsf{SOrd}^{\mathcal{H}}(F(\Delta^+), F(\Delta^+))$ as $T'(\sigma)$. Because we only work with free algebras, we can uniquely specify $T'(\sigma)$ by considering $\overline{\sigma} = T'(\sigma) \circ \eta_{\Delta^+} \in \mathsf{SOrd}(\Delta^+, \mathcal{H}(\Delta^+))$:

$$\begin{aligned}
\overline{\sigma}(\iota_0(\diamond)) &\coloneqq \eta_{\Delta^+}(\iota_0(\diamond)) \\
\overline{\sigma}(\iota_1(\alpha)) &\coloneqq \mathcal{H}(\iota_1)(\sigma(\eta_{\Delta}(\alpha))) \\
\overline{\sigma}(\iota_2(\alpha)) &\coloneqq \eta_{\Delta^+}(\iota_2(\alpha))
\end{aligned}$$

Then, the $T'(\sigma)$ can be defined in terms of $\overline{\sigma}$ such that $\overline{\sigma} = T'(\sigma) \circ \eta_{\Delta^+}$:

$$T'(\sigma) \coloneqq \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma})$$

With the help of T', we are ready to define $T(\sigma) \in \mathsf{SOrd}^{\mathcal{M}_{\mathcal{D}}}(F(\Psi), F(\Psi))$ as follows:

$$T(\sigma)(\alpha, d) \coloneqq (\alpha, T'(\sigma) \bullet_{\mathcal{D}} d)$$

We need to show that $T(\sigma)$ preserves the strict order of $\mathcal{M}_{\mathcal{D}}$, which is generated by these two:

(1) $(\alpha, d_1) < (\alpha, d_2)$ for any $\alpha \in \Psi$ and any $d_1 < d_2 \in \mathcal{D}$.
(2) $(\alpha, \mathbf{1}_{F^{\mathcal{H}}(\Delta)}) < (\beta, \mathbf{1}_{F^{\mathcal{H}}(\Delta)})$ for any $\alpha < \beta \in \Psi$.

The first holds because $<_\Delta$ is left-invariant. The second holds vacuously because Ψ is discrete. (If Ψ is not discrete, then we would need to show $(\alpha, T'(\sigma)) < (\beta, T'(\sigma))$, which is in general false.)

[5] A set is a discrete poset.

Identities. Suppose $\sigma = \mathbf{id}_{F(\Delta)}$. We will consider $\overline{\mathbf{id}} = T'(\mathbf{id}) \circ \eta_{\Delta^+}$ that uniquely determines $T'(\mathbf{id})$:

$$\begin{aligned}\overline{\mathbf{id}} \circ \iota_0 &= \eta_{\Delta^+} \circ \iota_0\\ \overline{\mathbf{id}} \circ \iota_1 &= \mathcal{H}(\iota_1) \circ \eta_\Delta = \eta_{\Delta^+} \circ \iota_1\\ \overline{\mathbf{id}} \circ \iota_2 &= \eta_{\Delta^+} \circ \iota_2\end{aligned}$$

Therefore $\overline{\mathbf{id}} = \eta_{\Delta^+}$ and so $T'(\mathbf{id}) = \mathbf{id}$. Consequently $T(\mathbf{id}) = \mathbf{id}$ as well, because:

$$T(\mathbf{id})(\alpha, d) = (\alpha, T'(\mathbf{id}) \bullet_{\mathcal{D}} d) = (\alpha, \mathbf{id} \bullet_{\mathcal{D}} d) = (\alpha, d)$$

Composition. Suppose we have endomorphisms $\sigma, \delta \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta), F(\Delta))$. We want to show that $T(\delta \circ \sigma) = T(\delta) \circ T(\sigma)$. First, we expand both sides by their definitions:

$$\begin{aligned}T(\delta \circ \sigma)(\alpha, d) &= (\alpha, T'(\delta \circ \sigma) \bullet_{\mathcal{D}} d)\\ T(\delta)(T(\sigma)(\alpha, d)) &= (\alpha, T'(\delta) \bullet_{\mathcal{D}} T'(\sigma) \bullet_{\mathcal{D}} d) = (\alpha, (T'(\delta) \circ T'(\sigma)) \bullet_{\mathcal{D}} d)\end{aligned}$$

It suffices to show $T'(\delta \circ \sigma) = T'(\delta) \circ T'(\sigma)$. Note both sides are determined by their precomposition with the monad unit η_{Δ^+}. Consider an input $\alpha \in \Delta^+$ to the precompositions with η_{Δ^+}:

- For $\alpha \in \operatorname{Im} \iota_1$, it suffices to consider the further precomposition with ι_1:

$$\begin{aligned}T'(\delta \circ \sigma) \circ \eta_{\Delta^+} \circ \iota_1 &= \mathcal{H}(\iota_1) \circ \delta \circ \sigma \circ \eta_\Delta\\ T'(\delta) \circ T'(\sigma) \circ \eta_{\Delta^+} \circ \iota_1 &= T'(\delta) \circ \mathcal{H}(\iota_1) \circ \sigma \circ \eta_\Delta\\ &= \mu_{\Delta^+} \circ \mathcal{H}(\bar{\delta}) \circ \mathcal{H}(\iota_1) \circ \sigma \circ \eta_\Delta\\ &= \mu_{\Delta^+} \circ \mathcal{H}(\bar{\delta} \circ \iota_1) \circ \sigma \circ \eta_\Delta\\ &= \mu_{\Delta^+} \circ \mathcal{H}(\mathcal{H}(\iota_1) \circ \delta \circ \eta_\Delta) \circ \sigma \circ \eta_\Delta\\ &= \mathcal{H}(\iota_1) \circ \delta \circ \sigma \circ \eta_\Delta\end{aligned}$$

- Otherwise, α must be $\iota_0(\diamond)$ or $\iota_2(\alpha')$ for some $\alpha' \in \Delta$. In either case,

$$T'(\delta \circ \sigma)(\eta_{\Delta^+}(\alpha)) = \eta_{\Delta^+}(\alpha) = T'(\delta)(\eta_{\Delta^+}(\alpha)) = T'(\delta)(T'(\sigma)(\eta_{\Delta^+}(\alpha)))$$

Natural transformation. Let $\blacklozenge$ be a fixed element in Ψ. For $\ell \in \mathcal{H}(\Delta)$, we will define an element in $\mathcal{D}$ and thus an element in $\mathcal{M}_{\mathcal{D}}(\Psi)$. Elements in $\mathcal{D}$ are uniquely determined by their precomposition with η_{Δ^+}; similar to how we defined $T'(\sigma)$ via $\overline{\sigma}$, we first define $\bar{\ell} \in \mathrm{SOrd}(\Delta^+, \mathcal{H}(\Delta^+))$:

$$\begin{aligned}\bar{\ell}(\iota_0(\diamond)) &\coloneqq \mathcal{H}(\iota_1)(\ell)\\ \bar{\ell}(\iota_1(\alpha)) &\coloneqq \eta_{\Delta^+}(\iota_2(\alpha))\\ \bar{\ell}(\iota_2(\alpha)) &\coloneqq \eta_{\Delta^+}(\iota_2(\alpha))\end{aligned}$$

Then, the element in $\mathcal{D}$ we want is $\mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell})$. That is, the universe level ℓ is mapped to

$$\tilde{\ell} \coloneqq (\blacklozenge, \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell})) \in \mathcal{M}_{\mathcal{D}}(\Psi)$$

We claim the mapping $\nu : \ell \mapsto \tilde{\ell}$ forms a natural transformation from $U^{\mathcal{H}}$ to $U^{\mathcal{M}_{\mathcal{D}}} \circ T$:

- ν is $<$-preserving. Suppose $\ell' < \ell$. By construction, $\overline{\ell'}(\alpha) \le \bar{\ell}(\alpha)$ for all $\alpha \in \Delta^+$ and

$$\overline{\ell'}(\iota_0(\diamond)) = \ell' < \ell = \bar{\ell}(\iota_0(\diamond))$$

  Therefore, $\nu(\ell') = \tilde{\ell'} <_{\mathcal{D}} \tilde{\ell} = \nu(\ell')$.

- ν is natural. Suppose we have $\sigma \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta), F(\Delta))$.

$$\begin{aligned}
\nu(\sigma(\ell)) &= (\blacklozenge, \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma(\ell)})) \\
T(\sigma)(\nu(\ell)) &= T(\sigma)(\blacklozenge, \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell})) \\
&= (\blacklozenge, T'(\sigma) \circ \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell})) \\
&= (\blacklozenge, \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell}))
\end{aligned}$$

  It suffices to show that $\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma(\ell)}) = \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell}) \in \mathcal{D}$. Because these morphisms are between free $\mathcal{H}$-algebras, it suffices to compare their precompositions with η_{Δ^+}:

$$\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma(\ell)}) \circ \eta_{\Delta^+} = \overline{\sigma(\ell)}$$

$$\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \mu_{\Delta^+} \circ \mathcal{H}(\bar{\ell}) \circ \eta_{\Delta^+} = \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \bar{\ell}$$

  - For variables $\alpha = \iota_0(\diamond) \in \Delta^+$,

$$\begin{aligned}
\overline{\sigma(\ell)}(\iota_0(\diamond)) &= \mathcal{H}(\iota_1)(\sigma(\ell)) \\
(\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \bar{\ell})(\iota_0(\diamond)) &= (\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}))(\mathcal{H}(\iota_1)(\ell)) \\
&= (\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma} \circ \iota_1))(\ell) \\
&= (\mu_{\Delta^+} \circ \mathcal{H}(\mathcal{H}(\iota_1) \circ \sigma \circ \eta_{\Delta}))(\ell) \\
&= (\mathcal{H}(\iota_1) \circ \sigma)(\ell)
\end{aligned}$$

  - For variables $\alpha = \iota_1(\alpha') \in \Delta^+$ where $\alpha' \in \Delta$, consider the further precomposition with ι_1:

$$\overline{\sigma(\ell)} \circ \iota_1 = \eta_{\Delta^+} \circ \iota_2$$

$$\mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \bar{\ell} \circ \iota_1 = \mu_{\Delta^+} \circ \mathcal{H}(\overline{\sigma}) \circ \eta_{\Delta^+} \circ \iota_2 = \overline{\sigma} \circ \iota_2 = \eta_{\Delta^+} \circ \iota_2$$

    The intuition is that variables in the first copy of Δ have dodged into the second copy.
  - For variables $\alpha = \iota_2(\alpha') \in \Delta^+$ where $\alpha' \in \Delta$, the proof is similar to the above case. The intuition is that the second copy of Δ will never be affected by subsequent substitutions.

Faithfulness. For parallel $\sigma, \delta \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta), F(\Delta))$ such that $T(\sigma) = T(\delta)$, we have $T'(\sigma) = T'(\delta)$ as the second components of $T(-)$. Therefore,

$$\mathcal{H}(\iota_1) \circ \sigma \circ \eta_{\Delta} = T'(\sigma) \circ \eta_{\Delta^+} \circ \iota_1 = T'(\delta) \circ \eta_{\Delta^+} \circ \iota_1 = \mathcal{H}(\iota_1) \circ \delta \circ \eta_{\Delta}$$

If $\mathcal{H}$ preserves monos, $\mathcal{H}(\iota_1)$ is monic and $\sigma \circ \eta_{\Delta} = \delta \circ \eta_{\Delta}$; thus $\sigma = \delta$, which means T is faithful. □

With Lemma 3.8, what remains is the embedding of morphisms between small free $\mathcal{H}$-algebras and their elements into endomorphisms and elements of a large enough free $\mathcal{H}$-algebra. The following lemma does exactly that, where the larger algebra is essentially generated by $\coprod_i \Delta_i$, the disjoint coproduct of generators of small free $\mathcal{H}$-algebras. Similar to Lemma 3.8, embedding of endofunctors is phrased as a functor T and that of elements as a natural transformation ν:

$$\begin{array}{ccc}
F(\Delta_i) & \xrightarrow{\alpha} & F(\Delta_j) \\
\\
F(\Delta) & \xrightarrow{T(\alpha)} & F(\Delta)
\end{array}
\qquad\qquad
\begin{array}{ccc}
\mathcal{H}(\Delta_i) & \xrightarrow{U(\alpha)} & \mathcal{H}(\Delta_j) \\
\downarrow{\scriptstyle \nu_{F(\Delta_i)}} & & \downarrow{\scriptstyle \nu_{F(\Delta_j)}} \\
\mathcal{H}(\Delta) & \xrightarrow{U(T(\alpha))} & \mathcal{H}(\Delta)
\end{array}$$

Note that we will use the same "dodging" technique in Lemma 3.8 but with infinitely many copies of $\coprod_i \Delta_i$ instead of just two. In Lemma 3.8, we could afford the collision of the first and the

second copies of Δ when embedding elements as the second copy of Δ will never be used, but here, in order for the embedding of morphisms to be functorial (in particular, preserving isomorphisms), no collision can happen due to dodging. This leads to the scheme of using infinitely many copies $\coprod_{n\in\mathbb{N}} \coprod_i \Delta_i$ to embed morphisms without collision. As an analogy, the infinite dodging is similar to how guests can always make space for a new guest in Hilbert's infinite hotels by moving to their successor rooms. We also have to move those guests back to their original rooms (the predecessors) when the new guest checks out, so that the resulting map can be an isomorphism.

Lemma 3.9. *Let $(\mathcal{H}, \eta, \mu)$ be any hierarchy theory and $\{\Delta_i\}_i$ be a small collection of small posets. There is a functor T out of the full subcategory of $\mathrm{SOrd}^{\mathcal{H}}$ induced by the free algebras $\{F^{\mathcal{H}}(\Delta_i)\}_i$ and into the full subcategory of $\mathrm{SOrd}^{\mathcal{H}}$ with one object $F^{\mathcal{H}}(\coprod_{n\in\mathbb{N}} \coprod_i \Delta_i)$. Moreover, there is a natural transformation from $U^{\mathcal{H}}$ to $U^{\mathcal{H}} \circ T$, and if $\mathcal{H}$ preserves monos, T is faithful.*

Proof. Let $\Delta = \coprod_{n\in\mathbb{N}} \coprod_i \Delta_i$. The functor T must map all objects to $F(\Delta)$. For morphisms, suppose we have $\sigma \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta_i), F(\Delta_j))$. Because we only work with free $\mathcal{H}$-algebras, we can uniquely specify $T(\sigma)$ by considering $\overline{\sigma} = T(\sigma) \circ \eta_\Delta \in \mathrm{SOrd}(\Delta, \mathcal{H}(\Delta))$:

$$\overline{\sigma}(\alpha) \coloneqq \begin{cases} \mathcal{H}(\iota_0 \circ \iota_j)(\sigma(\eta_{\Delta_i}(\alpha'))) & \text{if } \alpha = \iota_0(\iota_i(\alpha')) \text{ for some } \alpha' \in \Delta_i \\ \eta_\Delta(\iota_n(\alpha')) & \text{if } \alpha = \iota_{n+1}(\alpha') \text{ for some } \alpha' \in \operatorname{Im}\iota_i \setminus \operatorname{Im}\iota_j \\ \eta_\Delta(\iota_{n+1}(\alpha')) & \text{if } \alpha = \iota_n(\alpha') \text{ for some } \alpha' \in \operatorname{Im}\iota_j \setminus \operatorname{Im}\iota_i \\ \eta_\Delta(\alpha) & \text{otherwise} \end{cases}$$

We shall show $\overline{\sigma}$ preserves the strict order so that $\overline{\sigma} \in \mathrm{SOrd}(\Delta, \mathcal{H}(\Delta))$. Suppose $\alpha < \beta \in \Delta$. Due to the disjointness of $\coprod_{n\in\mathbb{N}} \coprod_i \Delta_i$, the elements α and β must belong to the same case in the above definition. Thus, we only have to check whether every case locally preserves the strict order:

- If both α and β are in $\operatorname{Im}(\iota_0 \circ \iota_i)$, we know $\mathcal{H}(\iota_0)$, $\mathcal{H}(\iota_j)$, σ, and η_{Δ_i} are all $<$-preserving.
- For all other cases: η_Δ and ι_n are $<$-preserving.

Then $T(\sigma)$ can be defined in terms of $\overline{\sigma}$ such that $\overline{\sigma} = T(\sigma) \circ \eta_\Delta$:

$$T(\sigma) \coloneqq \mu_\Delta \circ \mathcal{H}(\overline{\sigma})$$

$T(\sigma)$ preserves the $\mathcal{H}$-action because the actions of free $\mathcal{H}$-algebras (namely μ_Δ) are natural in Δ.

Identities. Suppose $i = j$ and $\sigma = \mathbf{id}_{F(\Delta_i)}$. We consider $\overline{\mathbf{id}} = T(\mathbf{id}) \circ \eta_\Delta$, which uniquely determines $T(\mathbf{id})$. For $\alpha \in \operatorname{Im}(\iota_0 \circ \iota_i)$, it suffices to consider the precomposition with $\iota_0 \circ \iota_i$:

$$\overline{\mathbf{id}} \circ \iota_0 \circ \iota_i = \mathcal{H}(\iota_0 \circ \iota_i) \circ \mathbf{id}_{F(\Delta_i)} \circ \eta_{\Delta_i} = \eta_\Delta \circ \iota_0 \circ \iota_i$$

Otherwise, if $\alpha \notin \operatorname{Im}(\iota_0 \circ \iota_i)$, the function $\overline{\mathbf{id}}$ still agrees with η_Δ because $\operatorname{Im}\iota_i = \operatorname{Im}\iota_j$ and only the last case of the definition applies. We conclude $\overline{\mathbf{id}} = \eta_\Delta$ and thus $T(\mathbf{id}) = \mathbf{id}_\Delta$.

Composition. Suppose we have $\sigma \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta_i), F(\Delta_j))$ and $\delta \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta_j), F(\Delta_k))$. We will show $T(\delta \circ \sigma) = T(\delta) \circ T(\sigma)$, using repeatedly the fact that morphisms out of free $\mathcal{H}$-algebras are determined by their precomposition with η_Δ.

- For $\alpha \in \operatorname{Im}(\iota_0 \circ \iota_i)$, it suffices to consider the precomposition with $\iota_0 \circ \iota_i$:

$$\begin{aligned} T(\delta \circ \sigma) \circ \eta_\Delta \circ \iota_0 \circ \iota_i &= \mathcal{H}(\iota_0 \circ \iota_k) \circ \delta \circ \sigma \circ \eta_{\Delta_i} \\ T(\delta) \circ T(\sigma) \circ \eta_\Delta \circ \iota_0 \circ \iota_i &= T(\delta) \circ \mathcal{H}(\iota_0 \circ \iota_i) \circ \sigma \circ \eta_{\Delta_i} \\ &= \mu_\Delta \circ \mathcal{H}(\overline{\delta}) \circ \mathcal{H}(\iota_0 \circ \iota_j) \circ \sigma \circ \eta_{\Delta_i} \\ &= \mu_\Delta \circ \mathcal{H}(\overline{\delta} \circ \iota_0 \circ \iota_j) \circ \sigma \circ \eta_{\Delta_i} \\ &= \mu_\Delta \circ \mathcal{H}(\mathcal{H}(\iota_0 \circ \iota_k) \circ \delta \circ \eta_{\Delta_j}) \circ \sigma \circ \eta_{\Delta_i} \end{aligned}$$

$$= \mathcal{H}(\iota_0 \circ \iota_k) \circ \delta \circ \sigma \circ \eta_{\Delta_i}$$

- Let $\alpha = \iota_n(\alpha')$. Except for the above case, $T(\sigma)$, $T(\delta)$ and $T(\delta \circ \sigma)$ essentially only change the index n (for $\coprod_{n\in\mathbb{N}} \ldots$) depending on the membership of α' in $\operatorname{Im}\iota_i$, $\operatorname{Im}\iota_k$, and $\operatorname{Im}\iota_j$:

Membership			$\overline{\sigma}$	$\overline{\delta}$	$\overline{\delta \circ \sigma}$
$\alpha' \in \operatorname{Im}\iota_i$	$\alpha' \in \operatorname{Im}\iota_j$	$\alpha' \in \operatorname{Im}\iota_k$	$n+1 \mapsto n+1$	$n+1 \mapsto n+1$	$n+1 \mapsto n+1$
$\alpha' \in \operatorname{Im}\iota_i$	$\alpha' \in \operatorname{Im}\iota_j$	$\alpha' \notin \operatorname{Im}\iota_k$	$n+1 \mapsto n+1$	$n+1 \mapsto n$	$n+1 \mapsto n$
$\alpha' \in \operatorname{Im}\iota_i$	$\alpha' \notin \operatorname{Im}\iota_j$	$\alpha' \in \operatorname{Im}\iota_k$	$n+1 \mapsto n$	$n \mapsto n+1$	$n+1 \mapsto n+1$
$\alpha' \in \operatorname{Im}\iota_i$	$\alpha' \notin \operatorname{Im}\iota_j$	$\alpha' \notin \operatorname{Im}\iota_k$	$n+1 \mapsto n$	$n \mapsto n$	$n+1 \mapsto n$
$\alpha' \notin \operatorname{Im}\iota_i$	$\alpha' \in \operatorname{Im}\iota_j$	$\alpha' \in \operatorname{Im}\iota_k$	$n \mapsto n+1$	$n+1 \mapsto n+1$	$n \mapsto n+1$
$\alpha' \notin \operatorname{Im}\iota_i$	$\alpha' \in \operatorname{Im}\iota_j$	$\alpha' \notin \operatorname{Im}\iota_k$	$n \mapsto n+1$	$n+1 \mapsto n$	$n \mapsto n$
$\alpha' \notin \operatorname{Im}\iota_i$	$\alpha' \notin \operatorname{Im}\iota_j$	$\alpha' \in \operatorname{Im}\iota_k$	$n \mapsto n$	$n \mapsto n+1$	$n \mapsto n+1$
$\alpha' \notin \operatorname{Im}\iota_i$	$\alpha' \notin \operatorname{Im}\iota_j$	$\alpha' \notin \operatorname{Im}\iota_k$	$n \mapsto n$	$n \mapsto n$	$n \mapsto n$

  For example, if $\alpha = \iota_{n+1}(\alpha')$ for some $\alpha' \in (\operatorname{Im}\iota_i \cap \operatorname{Im}\iota_k) \setminus \operatorname{Im}\iota_j$ (which is the third row),

$$T(\delta \circ \sigma)(\eta_\Delta(\iota_{n+1}(\alpha'))) = \eta_\Delta(\iota_{n+1}(\alpha')) = T(\delta)(\eta_\Delta(\iota_n(\alpha'))) = T(\delta)(T(\sigma)(\eta_\Delta(\iota_{n+1}(\alpha'))))$$

Natural transformation. The maps $\mathcal{H}(\iota_0 \circ \iota_i)$ form a natural transformation from $U^{\mathcal{H}}$ to $U^{\mathcal{H}} \circ T$ because, for any $\sigma \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta_i), F(\Delta_j))$,

$$\begin{aligned} T(\sigma) \circ \mathcal{H}(\iota_0 \circ \iota_i) &= \mu_\Delta \circ \mathcal{H}(\overline{\sigma}) \circ \mathcal{H}(\iota_0 \circ \iota_i) \\ &= \mu_\Delta \circ \mathcal{H}(\overline{\sigma} \circ \iota_0 \circ \iota_i) \\ &= \mu_\Delta \circ \mathcal{H}(\mathcal{H}(\iota_0 \circ \iota_j) \circ \sigma \circ \eta_{\Delta_i}) \\ &= \mathcal{H}(\iota_0 \circ \iota_j) \circ \sigma \end{aligned}$$

Faithfulness. For parallel $\sigma, \delta \in \mathrm{SOrd}^{\mathcal{H}}(F(\Delta_i), F(\Delta_j))$ such that $T(\sigma) = T(\delta)$,

$$\mathcal{H}(\iota_0 \circ \iota_j) \circ \sigma \circ \eta_{\Delta_i} = T(\sigma) \circ \eta_\Delta \circ \iota_0 \circ \iota_i = T(\delta) \circ \eta_\Delta \circ \iota_0 \circ \iota_i = \mathcal{H}(\iota_0 \circ \iota_j) \circ \delta \circ \eta_{\Delta_i}$$

If $\mathcal{H}$ preserves monos, $\mathcal{H}(\iota_0 \circ \iota_j)$ is monic and $\sigma \circ \eta_{\Delta_i} = \delta \circ \eta_{\Delta_i}$ and $\sigma = \delta$; thus T is faithful. □

Remark. *We have optimized the formulations of Lemmas 3.8 and 3.9 for comprehensibility; if one instead wishes to reduce the size of $\mathcal{D}$, there are at least two possible modifications one can make:*

- *The functor in Lemma 3.9 was carefully constructed so that it works even if* $\operatorname{Im}\iota_i$ *and* $\operatorname{Im}\iota_j$ *overlap, so long as every Δ_i is a* decidable subobject[6] *of some fixed Δ. We simply chose $\Delta = \coprod_i \Delta_i$ to avoid discussing decidable subobjects in the statement of the lemma.*
- *It is possible to avoid the second copy of Δ in Lemma 3.8 by recycling the infinite buffer in Lemma 3.9 (that is, $\coprod_{n\in\mathbb{N}} \coprod_i \Delta_i$). One can use the odd layers as the first copy and the even layers as the second. This would lead to a more economic embedding, at the cost of clean lemmas.*

Remark. *One might wonder if we can replace $\{\diamond\} \amalg \Delta \amalg \Delta$ in the proof of Lemma 3.8 with our* constant displacements *(Section 3.3.2), treating universe levels as constants and substitution as a right-action of endomorphisms on those levels. Unfortunately this does not work because substitution is not a right-action of endomorphisms: it preserves the strict order of levels,* not *of endomorphisms.*

Theorem 3.10. *Let $(\mathcal{H}, \eta, \mu)$ be any hierarchy theory, $\{\Delta_i\}_i$ be a small collection of small posets, and Ψ be any non-empty set. There exists a displacement algebra $\mathcal{D}$ such that there is a functor T out of the full subcategory of* $\mathrm{SOrd}^{\mathcal{H}}$ *induced by the free algebras $\{F^{\mathcal{H}}(\Delta_i)\}_i$ and into the full subcategory*

[6] A subobject S of V is decidable if V is isomorphic to the coproduct of S and its complement.

of $\mathrm{SOrd}^{\mathcal{M}_\mathcal{D}}$ *with one object* $F^{\mathcal{M}_\mathcal{D}}(\Psi)$. *Moreover, there is a natural transformation from* $U^\mathcal{H}$ *to* $U^{\mathcal{M}_\mathcal{D}} \circ T$, *and if* $\mathcal{H}$ *preserves monos,* T *is faithful.*

Proof. Apply Lemma 3.9 to obtain a functor T_1 and a natural transformation $\nu : U^\mathcal{H} \to U^\mathcal{H} \circ T_1$. Choose any singleton poset $\{\diamond\}$. Apply Lemma 3.8 with $\{\diamond\}$ and $\Delta = \coprod_{n\in\mathbb{N}} \coprod_i \Delta_i$ to obtain a functor T_2 and a natural transformation $\xi : U^\mathcal{H} \to U^{\mathcal{M}_\mathcal{D}} \circ T_2$. Let the functor T be $T_2 \circ T_1$ and the natural transformation be $\xi_{T_1(-)} \circ \nu$. T is faithful if both T_1 and T_2 are faithful. See the following commuting diagram that puts T_1, T_2, ν, and ξ together:

$$\begin{array}{ccc}
F^\mathcal{H}(\Delta_i) & \xrightarrow{\alpha} & F^\mathcal{H}(\Delta_j) \\
F^\mathcal{H}(\Delta) & \xrightarrow{T_1(\alpha)} & F^\mathcal{H}(\Delta) \\
F^{\mathcal{M}_\mathcal{D}}(\Psi) & \xrightarrow{T_2(T_1(\alpha))} & F^{\mathcal{M}_\mathcal{D}}(\Psi)
\end{array}
\qquad
\begin{array}{ccc}
\mathcal{H}(\Delta_i) & \xrightarrow{U(\alpha)} & \mathcal{H}(\Delta_j) \\
\downarrow{\scriptstyle \nu_{F(\Delta_i)}} & & \downarrow{\scriptstyle \nu_{F(\Delta_j)}} \\
\mathcal{H}(\Delta) & \xrightarrow{U(T_1(\alpha))} & \mathcal{H}(\Delta) \\
\downarrow{\scriptstyle \xi_{F(\Delta)}} & & \downarrow{\scriptstyle \xi_{F(\Delta)}} \\
\mathcal{M}_\mathcal{D}(\Psi) & \xrightarrow{U(T_2(T_1(\alpha)))} & \mathcal{M}_\mathcal{D}(\Psi)
\end{array}$$

□

Theorem 3.10 states that any hierarchy theory $\mathcal{H}$ induces a displacement algebra $\mathcal{D}$ such that $\mathcal{H}$ embeds into the McBride monad $\mathcal{M}_\mathcal{D}$: in other words, any universe-polymorphic type theory (in the sense of our paper) can be seen as an instance of generalized "crude but effective stratification," subject to the technical conditions above. In particular, we only consider free $\mathcal{H}$-algebras, which do not support contexts containing inequalities between non-variable level expressions (see Page 9).

4 OCAML IMPLEMENTATION AND AGDA MECHANIZATION

We have implemented an OCaml library mugen [RedPRL Development Team 2022a] for displacement algebras, intended for use by anyone who wishes to adopt our generalization of McBride's "crude but effective stratification" in their type checker or proof assistant. Our library provides a signature for displacement algebras and implements the derived level expressions and their comparators for use in a type checker. Users may either provide their own displacement algebras or use examples mentioned in Section 3.3. We are currently using our library in the development of our experimental proof assistant `algaett` [RedPRL Development Team 2022b]. The mugen library allows us to swap the underlying displacement algebra of `algaett` with very minimal changes to the definition of its core language, parser, pretty-printer, and elaboration of levels; in particular, we do *not* edit the type checker. This indicates that we have achieved a modular design.

We also have a Cubical Agda formalization of our construction of the generalized McBride monad $\mathcal{M}_\mathcal{D}$ (Section 3.1); the fact that the examples in Section 3.3 satisfy the axioms of displacement algebras and certain additional properties (bottom elements, joins, etc.); and a partial mechanization of Section 3.4. Our Cubical Agda formalization is available at https://github.com/RedPRL/agda-mugen, and is built atop the 1Lab [The 1Lab Development Team 2022].

5 DISCUSSION

In this paper, we have presented a generic framework for universe hierarchies and universe polymorphism, and proven that a generalization of McBride's "crude but effective stratification" is sufficiently expressive to capture all systems of universe polymorphism in our sense.

We are far from the first to note that universe hierarchies can be indexed by more general orders than the natural numbers [Huet 1987; Kovács 2022; McBride 2002]; our universe-monomorphic type

theory (Section 2.1) simply codifies this observation, with the added generality that the hierarchy need not be well-founded. However, our description of universe polymorphism in terms of monads on SOrd (Section 2.2) seems novel, and recasting McBride's "crude but effective stratification" in monads helped us identify a vast design space for universe hierarchies that includes exotic universe levels such as negative numbers, rational numbers, computational reals, and even strings.

Many researchers have attempted to address the usability of universe hierarchies: the "hands-off" approach of typical ambiguity was pioneered by Huet [1987] and Harper and Pollack [1991] and implemented in systems such as LEGO, Coq, and Idris. Other systems, such as Agda, Lean, and Matita, adopt more explicit forms of universe polymorphism. And other authors have considered intermediate points in the design space; for example, Courant [2002] considers a system in which one can hypothesize level variables $\alpha \geq \ell$ bounded by arbitrary level expressions ℓ. A summary of existing systems can be found in Kovács [2022, Section 6].

McBride introduced his "crude but effective stratification" out of frustration with existing systems, in terms of both usability and implementability. His scheme belongs to the second category—it requires users to write explicit levels—but with minimal syntactic burden. In other words, it avoids constraint solving altogether without forcing users to write complicated universe level expressions.[7] It is thus one of the easiest ways to support universe polymorphism, and we have shown that it is also universal. Experience using our own library suggests that the simplicity of "crude but effective stratification" has not been lost despite our generalization: we have built type checkers parametrized by arbitrary displacement algebras, and the additional complexity of displacement algebras can be completely outsourced to a dedicated library.

5.1 Future Work

Internalized level types and higher-ranked universe polymorphism. We believe there is limited need for internalized universe levels[8] or higher-ranked universe polymorphism for a *cumulative* universe hierarchy. Moreover, a stratification between levels and regular terms should simplify the implementation. (The notable example Agda does not have a fully cumulative hierarchy.) However, it remains theoretically interesting to internalize these levels or consider higher-ranked polymorphism. Recently, Kovács [2022] has constructed an inductive-recursive model of type theory with internalized universe levels and Bezem et al. [2022] has given an account of higher-ranked polymorphism. Both support user constraints between universe levels.

Non-free $\mathcal{H}$-algebras. As we noted in Section 2.2, we could consider type theories indexed by non-free $\mathcal{H}$-algebras, which would account for level contexts with strict inequalities between level *expressions*, not just level variables. Practically, this allows us to lift definitions in $\mathfrak{U}_{\alpha \vee \beta}$ to some $\mathfrak{U}_{1+\gamma}$, and hence to remove universe indices from the syntax by moving them into the constraints.

Sozeau and Tabareau [2014] have explored a similar system but only with constraints of the form $(\alpha_1 \vee \cdots \vee \alpha_n) < \beta$; it is not immediately obvious which other fragments admit efficient type checking and substitution algorithms. Bezem et al. [2008] describes an algorithm to handle constraints of the form $(\alpha \vee \beta) + n \geq \gamma$ over integers, and Bezem and Coquand [2022] present an algorithm for join semilattices with successors. These advances in algorithms suggest different fragments of non-free $\mathcal{H}$-algebras that can nevertheless be implemented efficiently.

[7] Note that an implementer may choose a complicated displacement algebra with a constraint solver to determine the order, but this is not required by McBride's system.

[8] One notable application of internalized universe levels is mechanized metatheory of type theories with universe hierarchies.

Partial displacement algebras. One might consider *partial* displacement algebras whose binary composition $\bullet$ is not always defined. This would allow for *indexed* displacements for which composition is defined only on elements with matching indices, in much the same way that groups can be generalized to groupoids. This could lead to a much simpler embedding of small $\mathcal{H}$-algebras into a displacement algebra because heterogeneous posets do not have to be embedded into a common one. However, we might lose the simplicity of our monadic formulation of hierarchy theories.

Large universes. People have considered further extensions of the universe hierarchy by super universes [Palmgren 1998], Mahlo universes [Setzer 2000], and higher-order universe operators [Takahashi 2022]. It is unclear how to integrate these into our framework.

Categorical semantics. Recently, Kovács [2022] developed a notion of *family diagram* to describe poset-indexed universe hierarchies in categories with families [Dybjer 1996], a categorical framework for the semantics of dependent type theory. We expect that we could define the categorical semantics of our universe-monomorphic type theory in terms of these family diagrams; one potential benefit would be a streamlined presentation of our universe-polymorphic type theory in terms of hierarchy-preserving morphisms of natural models [Awodey 2018; Newstead 2018].

Separating universe lifts and universe membership. In this paper we model universe hierarchies as a single poset $(L, \leq)$ whose *non-strict* order $\ell \leq \ell'$ determines which lifts $\uparrow_\ell^{\ell'} : \mathfrak{U}_\ell \to \mathfrak{U}_{\ell'}$ exist between universes, and whose associated *strict* order $\ell < \ell'$ (defined as $\ell \leq \ell'$ and $\ell \neq \ell'$) determines when a universe is an element of another universe $\mathfrak{U}_\ell : \mathfrak{U}_{\ell'}$. As we discussed in Example 2.15, this prevents us from modeling the level-maximum operations of Lean and Agda, in which $\alpha \leq \alpha \vee \beta$ for the purposes of lifts, but neither $\alpha < \alpha \vee \beta$ nor $\alpha = \alpha \vee \beta$ for the purposes of membership.

Specifically, because we require level substitutions to preserve $<$, whenever $\ell \leq \ell'$ then we either have $\ell = \ell'$ or we have $\ell < \ell'$ under all substitutions. One solution is to define $\leq$ (lifts) and $<$ (membership) as two distinct relations with the property that $\ell < \ell' \leq \ell''$ implies $\ell < \ell''$ (if $\mathfrak{U}_\ell : \mathfrak{U}_{\ell'} \subseteq \mathfrak{U}_{\ell''}$ then $\mathfrak{U}_\ell : \mathfrak{U}_{\ell''}$). After decoupling lifts and membership, we can ask for level substitutions to (separately) preserve both relations, and then define $\vee$ as the join with respect to $\leq$.

Kinds of universes. Many proof assistants are equipped with either multiple parallel universe hierarchies or special impredicative universes. Examples include the strict propositions (`sProp`) in Agda, Coq, and Lean; the universe kinds in RedPRL [Angiuli et al. 2018]; and the support for two-level type theory [Annenkov et al. 2017] recently added to Agda. Levels differ from kinds in that type formers typically behave uniformly across universe levels, but non-uniformly with respect to universe kinds. For example, whether Π-types are strict propositions depends on their codomain only: if $A : \mathfrak{U}_\ell$ and $x{:}A \vdash B : \mathtt{sProp}_{\ell'}$ then $\prod_{x:A} B : \mathtt{sProp}_{\ell \vee \ell'}$, but if $A : \mathtt{sProp}_\ell$ and $x{:}A \vdash B : \mathfrak{U}_{\ell'}$ then $\prod_{x:A} B : \mathfrak{U}_{\ell \vee \ell'}$. Our current framework is not well-equipped to handle universe kinds, but we believe it would be possible to account for these features if we decoupled the lifting and membership relations as described above.

ACKNOWLEDGMENTS

We thank Michael Shulman and Jonathan Sterling for helpful conversations about hierarchy theories. We thank Jad Ghalayini for suggesting fractal universe levels (Section 3.3.7) to us during WITS 2022 (Workshop on the Implementation of Type Systems), and Steve Awodey for suggesting integral universe levels (Example 2.3).

This research was sponsored by the U.S. Air Force Office of Scientific Research under grant number FA9550-21-0009. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

REFERENCES

Carlo Angiuli, Evan Cavallo, Kuen-Bang Hou (Favonia), Robert Harper, and Jonathan Sterling. 2018. The RedPRL Proof Assistant (Invited Paper). In Proceedings of the 13th International Workshop on *Logical Frameworks and Meta-Languages: Theory and Practice,* Oxford, UK, 7th July 2018 *(Electronic Proceedings in Theoretical Computer Science)*, Frédéric Blanqui and Giselle Reis (Eds.), Vol. 274. Open Publishing Association, 1–10. https://doi.org/10.4204/EPTCS.274.1

Danil Annenkov, Paolo Capriotti, and Nicolai Kraus. 2017. Two-Level Type Theory and Applications. arXiv:cs.LO/1705.03307 http://arxiv.org/abs/1705.03307 Preprint.

Steve Awodey. 2018. Natural models of homotopy type theory. *Mathematical Structures in Computer Science* 28, 2 (2018), 241–286. https://doi.org/10.1017/S0960129516000268

Marc Bezem and Thierry Coquand. 2022. Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism. *Theoretical Computer Science* 913 (2022), 1–7. https://doi.org/10.1016/j.tcs.2022.01.017

Marc Bezem, Thierry Coquand, Peter Dybjer, and Martín Escardó. 2022. Type Theories with Universe Level Judgments. https://types22.inria.fr/files/2022/06/TYPES_2022_paper_56.pdf

Marc Bezem, Robert Nieuwenhuis, and Enric Rodríguez-Carbonell. 2008. The Max-Atom Problem and Its Relevance. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Iliano Cervesato, Helmut Veith, and Andrei Voronkov (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 47–61. https://doi.org/10.1007/978-3-540-89439-1_4

Edwin C. Brady. 2013. Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming* 23 (2013), 552 – 593. https://doi.org/10.1017/S095679681300018X

Thierry Coquand. 1986. An Analysis of Girard's Paradox. In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*. IEEE Computer Society Press, 227–236.

Thierry Coquand. 2013. Presheaf model of type theory. (2013). http://www.cse.chalmers.se/~coquand/presheaf.pdf Unpublished note.

Thierry Coquand. 2019. Canonicity and normalization for dependent type theory. *Theoretical Computer Science* 777 (2019), 184–191. https://doi.org/10.1016/j.tcs.2019.01.015 In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part I.

Judicaël Courant. 2002. Explicit Universes for the Calculus of Constructions. In *Theorem Proving in Higher Order Logics*, Victor A. Carreño, César A. Muñoz, and Sofiène Tahar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 115–130. https://doi.org/10.1007/3-540-45685-6_9

Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *Automated Deduction – CADE-25*, Amy P. Felty and Aart Middeldorp (Eds.). Springer International Publishing, Cham, 378–388. https://doi.org/10.1007/978-3-319-21401-6_26

Peter Dybjer. 1996. Internal type theory. In *Types for Proofs and Programs (TYPES 1995) (Lecture Notes in Computer Science)*, Stefano Berardi and Mario Coppo (Eds.), Vol. 1158. Springer Berlin Heidelberg, Berlin, Heidelberg, 120–134. https://doi.org/10.1007/3-540-61780-9_66

Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. 2019. Definitional Proof-Irrelevance without K. *Proc. ACM Program. Lang.* 3, POPL, Article 3 (jan 2019), 28 pages. https://doi.org/10.1145/3290316

Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Multimodal Dependent Type Theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '20)*. Association for Computing Machinery, New York, NY, USA, 492–506. https://doi.org/10.1145/3373718.3394736

Trang Ha and Valentina Harizanov. 2018. Orders on magmas and computability theory. *Journal of Knot Theory and Its Ramifications* 27, 07 (2018), 1841001. https://doi.org/10.1142/S0218216518410018

Robert Harper and Robert Pollack. 1991. Type checking with universes. *Theoretical Computer Science* 89, 1 (1991), 107–136. https://doi.org/10.1016/0304-3975(90)90108-T

Gérard Huet. 1987. Extending the calculus of constructions with Type:Type. (1987). http://pauillac.inria.fr/~huet/PUBLIC/typtyp.pdf Unpublished note.

András Kovács. 2022. Generalized Universe Hierarchies and First-Class Universe Levels. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022) (Leibniz International Proceedings in Informatics (LIPIcs))*, Florin Manea and Alex Simpson (Eds.), Vol. 216. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:17. https://doi.org/10.4230/LIPIcs.CSL.2022.28

Per Martin-Löf. 1971. An intuitionistic theory of types. (1971). Unpublished preprint.

Per Martin-Löf. 1975. An Intuitionistic Theory of Types: Predicative Part. In *Logic Colloquium '73*, H.E. Rose and J.C. Shepherdson (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 80. Elsevier, 73–118. https://doi.org/10.1016/S0049-237X(08)71945-1

Conor McBride. 2002. Crude but Effective Stratification. https://personal.cis.strath.ac.uk/conor.mcbride/Crude.pdf Slides.

Conor McBride. 2011. Crude but Effective Stratification. https://mazzo.li/epilogue/index.html%3Fp=857&cpage=1.html

Clive Newstead. 2018. *Algebraic models of dependent type theory.* Ph.D. Dissertation. Carnegie Mellon University. https://www.math.cmu.edu/~cnewstea/thesis-clive-newstead.pdf

Erik Palmgren. 1998. On universes in type theory. *Twenty five years of constructive type theory* (1998), 191–204. https://doi.org/10.1093/oso/9780198501275.001.0001

RedPRL Development Team. 2022a. mugen. https://github.com/RedPRL/mugen

RedPRL Development Team. 2022b. algaett. https://github.com/RedPRL/algaett

Anton Setzer. 2000. Extending Martin-Löf type theory by one Mahlo-universe. *Archive for Mathematical Logic* 39, 3 (2000), 155–181. https://doi.org/10.1007/s001530050140

Matthieu Sozeau and Nicolas Tabareau. 2014. Universe Polymorphism in Coq. In *Interactive Theorem Proving*, Gerwin Klein and Ruben Gamboa (Eds.). Springer International Publishing, Cham, 499–514. https://doi.org/10.1007/978-3-319-08970-6_32

Yuta Takahashi. 2022. Higher-Order Universe Operators in Martin-Löf Type Theory with one Mahlo Universe. https://types22.inria.fr/files/2022/06/TYPES_2022_paper_63.pdf

The 1Lab Development Team. 2022. The 1Lab. https://1lab.dev

The Agda Development Team. 2022. The Agda Programming Language. https://wiki.portal.chalmers.se/agda/pmwiki.php

The Coq Development Team. 2022. The Coq Proof Assistant. https://www.coq.inria.fr

The HELM Team. 2016. Matita. http://matita.cs.unibo.it/index.shtml

The LEGO Team. 1999. The LEGO Proof Assistant. https://www.dcs.ed.ac.uk/home/lego/

Received 2022-07-07; accepted 2022-11-07